UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR
DEP. TECNOLOGÍA ELECTRÓNICA

SISTEMA DE CONTROL Y MONITORIZACIÓN DE UN RACK VME

PROYECTO FINAL DE CARRERA
INGENIERÍA TÉCNICA INDUSTRIAL
ELECTRÓNICA INDUSTRIAL

24 Septiembre 2015

Autor:
Claudio Jiménez Castillo
Tutor:
Pedro Contreras Lallana

A mi familia y de manera muy especial a mis padres Jorge y Dora, por el apoyo y la paciencia que han tenido conmigo a lo largo de toda mi vida.

Contenido

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

1.1. Introducción

El VERSAmodule Eurocard bus, más conocido como VME o VMEbus, es un bus informático desarrollado en primera instancia por Motorola y que se estandarizó como norma IEEE en el año 1987 [IEE02]. Principalmente es utilizado para crear sistemas integrados y en tiempo real y aunque está ampliamente extendido en sistemas militares, industriales y aeroespaciales, fuera de estos, su uso es reducido.

Se trata de un sistema modular formado por un chasis[1], placas de procesado de datos y placas de adquisicion o emisión de señal que se utilizan como equipos embarcados de control en diversos vehículos, como pueden ser barcos, helicopteros o aviones, tanto militares como civiles.

Este tipo de uso suele requerir que los contratos de mantenimiento de estos equipos se prolonguen durante muchos años, lo que también los hace atractivos para un sector como el aeroespacial, donde los proyectos y misiones pueden estar activos durante décadas.

[1] Los chasis de los sistemas VME son, generalmente, subchasis enracables o preparados a medida para poder integrarlos facilmente donde van a ser instalados.

Este tipo de equipos suele usarse dentro de sistemas certificados como *Safety of Life*[2], por lo que es primordial que el hardware se pueda monitorizar y controlar.

Por este motivo, el propósito de este proyecto es diseñar y crear el prototipo de un sistema de monitorización y control para un subchasis VME que regule su temperatura para que ésta no se eleve por encima de los límites de funcionamiento de los componentes que se instalen en dicho chasis. También deberá informar cuando los niveles de tensión proporcionados por la fuente de alimentación estén fuera de los límites aceptables de trabajo.

Este sistema va a estar formado por una placa tipo PCB y un programa de ordenador. La placa será autónoma y se encargará de monitorizar constantemente la temperatura y los niveles de tensión proporcionados por la fuente de alimentación, así como de controlar los ventiladores y los LEDs de estado del sistema.

El programa de ordenador va a ser desarrollado para correr sobre un sistema Linux y se conectará a la placa a través de un cable USB. Mediante este software se podrá recoger la información de la placa, que se mostrará por pantalla y almacenará en disco para su posterior análisis si así se le indica.

1.2. Objetivos

El objetivo fundamental de este proyecto es diseñar un equipo lo más robusto posible que monitorice los niveles de tensión entregados por una fuente de alimentación y la temperatura dentro de un chasis VME. A partir de estos datos, controlará unos LEDs de alerta y unos ventiladores para mantener la temperatura dentro de unos niveles aceptables.

También se creará un programa de ordenador que se comunique con la placa desarrollada, recoja la información obtenida por dicha placa y la muestre al operador, con la opción de guardarla por si es necesario un analisis posterior de estos datos.

Para aumentar en todo lo posible la robustez del sistema, se optara por elegir unos componentes que estén suficientemente probados y minimizar la complejidad del sistema en la medida de lo posible para disminuir los posibles puntos de fallo. Se diseñará como un sistema modular, que permita la modificación del diseño de manera sencilla para monitorizar cualquier modelo de chasis VME u otros sistemas electrónicos. También debería ser factible utilizar otros modelos de sensores, lo que permitiría reutilizar otros sensores de temperatura ya instalados.

[2]Los sistemas o servicios *Safety of Life* son aquellos de los que dependen vidas humanas

1.3. Estructura de la memoria

La memoria se ha estructurado de la siguiente manera:

Capítulo 1: Introducción y objetivos. Incluye una breve introducción, los objetivos a cumplir en este proyecto, este apartado y las fases de desarrollo del proyecto.

Capítulo 2: Estudio previo. Estudio previo donde se discuten los métodos elegidos para medir temperatura y voltajes, la topología con la que se va a diseñar la placa y los lenguajes que se van a utilizar para implementar los programas.

Capítulo 3: Diseño de la placa de control. Cálculos teóricos y diseño de la placa de control. También se incluye una estimación de los costes de los materiales que se utilizarán en la fabricación de un prototipo.

Capítulo 4: Comunicaciones. En este capítulo se detallará el protocolo de comunicaciones que van a utilizar el programa de ordenador y la placa para mandarse información.

Capítulo 5: Programas. Este capítulo documenta el proceso de diseño e implementación del código del microcontrolador y del programa para mantenimiento. Detallando todas las funciones de los programas y que hace cada una de ellas.

Apéndice A: Componentes. Relación de los componentes finalmente utilizados para realizar el prototipo y sus características más significativas. También incluye un listado con los enlaces a sus hojas de características.

Apéndice B: Código fuente. Código fuente de los programas creados en este proyecto.

1.4. Fases de desarrollo

Se ha dividido el proyecto en siete apartados para su implementación:

- Selección de los componentes adecuados y suficientemente probados para asegurar la robustez del sistema.

- Desarrollo del programa para ordenador siguiendo las especificaciones descritas en su apartado, ha de poder comunicarse con la placa para leer los valores mostrándolos por pantalla y permitir su almacenamiento para su posterior análisis.
- Diseño e implementación del hardware para la medición de las tensiones y las temperaturas, y el control de los ventiladores y de los LEDs de alerta.
- Fabricación de la placa.
- Desarrollo del código que se ejecutará en el microcontrolador de la placa.
- Integración de las partes hardware y software del proyecto.
- Prueba del sistema completo para comprobar su correcto funcionamiento una vez integrado.

Capítulo 2

Estudio previo

En este capítulo se elegirán el modelo y el fabricante de los componentes principales del proyecto, incluida la placa del microcontrolador, lenguajes de programación a usar,... También se explica en líneas generales como se va a diseñar la placa y el programa de ordenador.

2.1. Placa

La placa es el componente principal del proyecto, es la encargada de adquirir las señales de los sensores, tratar dichas señales si es necesario, procesar la información y actuar sobre los LEDs de alerta y los ventiladores.

Para facilitar su diseño y sus posibles evoluciones posteriores se va a dividir en tres bloques: bloque de adquisición de señales, bloque de control y comunicaciones y bloque de actuadores. De esta forma, por ejemplo, si en el futuro se quiere cambiar un modulo específico, se podrá modificar sólo su bloque sin que esto afecte al diseño del resto de la placa. Sólo habrá que mantener el comportamiento externo del bloque.

Se puede ver un diagrama de bloques inicial de la placa en la figura 2.1.

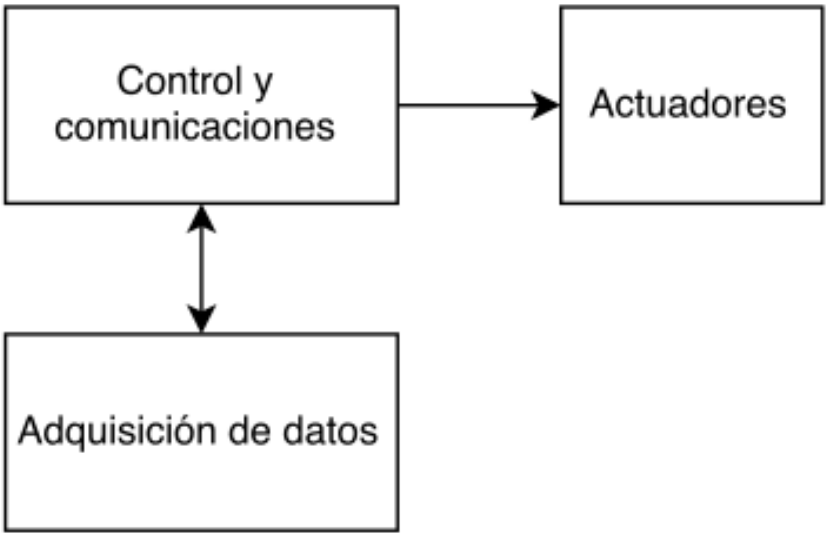

Figura 2.1: Diagrama de bloques inicial de la placa.

2.1.1. Adquisición de datos

Este bloque se encarga de transformar los valores dados por los sensores de temperatura y de tensión para que puedan ser leídos correctamente por el convertidor analógico/digital del microcontrolador.

Sensores de temperatura

El control de temperatura es un punto crítico en cualquier sistema electrónico. Trabajar a una temperatura superior a la temperatura máxima de funcionamiento expecificada por el fabricante, durante un período de tiempo prolongado, puede ocasionar un fallo del sistema y acelerar el deterioro de sus componentes. Como los límites de funcionamiento que se dan, siempre tienen un margen de seguridad, y como normalmente tampoco se trabajará en zonas límite de estos intervalos, se pueden asumir errores de uno o dos grados centigrados en las medidas sin que esto suponga un problema grave.

Teniendo en esto en cuenta, se ha decidido utilizar un sensor LM335 fabricado por STMicroelectronics con un intervalo de funcionamiento de entre -40 y +125 grados centígrados, suficiente para cualquier sistema electrónico convencional. En la tabla B.7 (página 59) se encuentran los datos específicos del modelo seleccionado.

Sensores de tensión

Este bloque se encarga de adaptar los valores de voltaje a medir ($+3,3\ V$, $+5\ V$, $+12\ V$ y $-12\ V$) para que estén dentro del rango de valores que puede leer el microcontrolador.

Para los valores positivos se utilizarán divisores de tensión formados por resistencias, y para el valor de $-12\ V$, se utilizará un amplificador con una topología inversora y una ganancía menor a 1.

2.1.2. Control

El elemento principal de esta sección es un microcontrolador. Se ha decidido utilizar una placa STM32F3Discovery, que viene con microcontrolador y varios de los elementos que necesitamos incorporados. Se trata de una placa altamente probada y de un fabricante de garantias como es STMicroelectronics. Esto asegura que el sistema tiene la robustez necesaria para entornos críticos.

La placa STM32F3Discovery cuenta con un microcontrolador ARM Cortex-M4 a 72MHz con 48 KB de RAM y una memoria de 256 KB en formato FLASH, lo que otorga suficiente memoria y potencia de cálculo para cubrir las necesidades de este proyecto. Sólo dispone de 4 conversores A/D de 12-bit, por lo que se va a colocar un multiplexor analógico para poder leer todos los sensores disponibles. Además, de esta forma, se asegura una posible ampliación del número de señales de entrada aumentando los pines utilizados para el control del multiplexor y cambiando éste chip por uno de más entradas, nos dará un aumento exponencial en el número de entradas.

Teniendo en cuenta que se van a monitorizar cuatro voltajes diferentes y al menos un sensor de temperatura, el multiplexor analógico ha de tener ocho entradas y una tensión de alimentación mayor a $3,3\ V$, ya que esta es la tensión máxima leída por el conversor A/D. Por estos motivos se ha elegido utilizar el multiplexor analógico 8x1 fabricado por Analog Devices modelo DG408BNZ cuyas características cumplen estas necesidades. Los datos específicos de este chip se encuentran en la tabla B.4 (página 58).

2.1.3. Actuadores

Sistema de alertas

Este bloque se encargará de encender los LEDs de estado del sistema. Para mantener la robusted y sencillez requeridas en este tipo de sistemas se va a optar por un circuito sencillo, utilizando siete bits de un puerto de salida conectando una resistencia y a continuación el LED.

Ventiladores

Los ventiladores se van a controlar mediante una señal PWM de la placa con una etapa amplificadora posterior. Esta etapa se encargará de adecuar el nivel de tensión de la señal de salida de la placa de control al utilizado por los ventiladores y estará compuesta por un amplificador colocado en modo no inversor y una ganancia mayor a 1.

2.2. Programas

Este proyecto cuenta con dos programas, uno principal en el microcontrolador y un segundo para ser usado desde un ordenador que se utilizará en tareas de mantenimiento y que se desarrollará sobre un sistema operativo GNU/Linux Debian para poder ejecutarse en cualquier sistema operativo Linux que tenga instaladas ciertas dependencias.

La placa de control elegida posee capacidad suficiente para el desarrollo del proyecto. Por este motivo se va a elegir C como lenguaje de programación, ya que aunque potencialmente el número de instrucciones sea mayor -penalizando un poco el rendimiento frente a otros lenguajes como puede ser Ensamblador- facilitará enormemente el desarrollo y posterior mantenimiento del código. También será mucho más sencillo realizar un cambio de placa si fuese necesario.

C también es un lenguaje ampliamente utilizado en la programación para ordenadores y en Linux permite el manejo de puertos USB. Teniendo en cuenta que ya se va a utilizar en la programación del microcontrolador, se ha decidido que la mejor opción es escribir este programa en el mismo lenguaje. Para la interfaz se va a utilizar Gtk, utilizada por el escritorio Gnome, uno de los escritorios más extendidos dentro del mundo del software libre. El desarrollo de esta interfaz se hará en Glade, uno de los distintos entornos de trabajo gráficos de desarrollo de Gtk que cubre

todas las necesidades que tendrá este proyecto.

Una de las premisas del programa de ordenador es que guarde los datos que recibe del microcontrolador para que puedan ser analizados posteriormente. Por esta razón y para que el programa pueda seguir siendo utilizado a la vez que guarda los datos, se ha elegido utilizar dos hilos. Mientras que el hilo padre se encarga del entorno gráfico del programa, el hijo se encarga de las comunicaciones con el microcontrolador y de guardar los datos en un fichero.

Capítulo 3

Diseño de la placa de control

El prototipo de la placa de control se creará en una placa de matriz de cara única de 1,5 mm de grosor, con orificios de 1 mm y paso 2,54 $\times$ 2,54 mm. Aunque todo se va a construir junto en una misma placa, para simplificar la fase de diseño y facilitar posibles modificaciones posteriores, se ha dividido en cuatro etapas funcionales. La primera y la segunda se encargarán de la adquisición de los datos de los niveles de tensión y de temperatura, mientras la tercera será la placa STM32F3Discovery que contiene el microcontrolador y se encarga del control, y la cuarta contendrá los actuadores de la fase de control.

3.1. Adquisición de datos

Esta primera parte se trata de una etapa de instrumentación que se encarga de la adquisición y acondicionamiento de las señales de los sensores de temperatura y de los voltajes para que puedan ser leídos por el microcontrolador. El número de dichas señales es suficientemente elevado como para que sea preciso el uso de un multiplexor para de cara a redcir el número de pines del multiplexor empleados. Utilizar este método facilitará aumentar el número de sensores si fuese necesario en un futuro, sin que ello significase tener que cambiar el microcontrolador, ya que

un aumento lineal en el número de pines, significa un aumento exponencial en el número de entradas.

Se ha elegido un multiplexor de la compañía Analog Devices modelo ADG408BNZ con una arquitectura ocho a una que cuenta con 16 pines. La configuración de los pines se puede encontrar en la tabla B.6 (página 58).

3.1.1. Tratamiento de los voltajes

La placa del microcontrolador necesita que los voltajes estén entre $0\,V$ y $+3\,V$, por lo que se ha diseñado un divisor de tensión para cada voltaje positivo y un amplificador operacional en configuración inversora y ganacia menor a 1 para la lectura de $-12\,V$. De esta, forma cuando los niveles de tensión estén en sus valores nominales, los datos que recibe el microcontrolador estarán alrededor de $2\,V$, pudiendo así detectar tanto aumentos como disminuciones en los voltajes. Los esquemas de estos divisores de tensión y de la configuración del amplificador están dibujados en la figura 3.1.

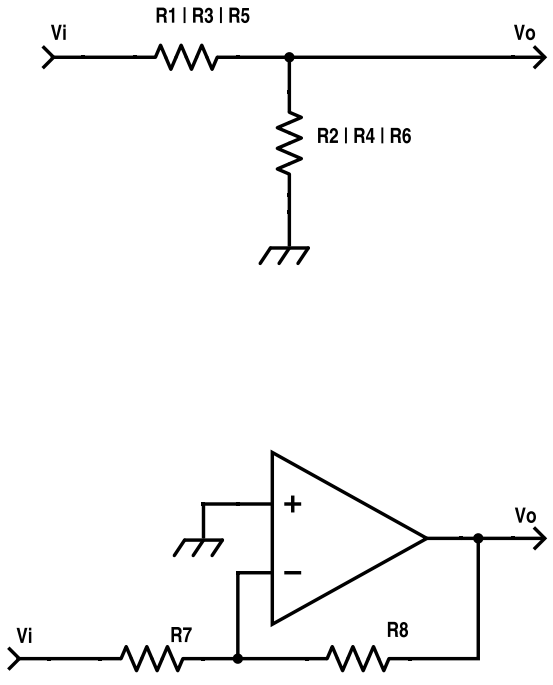

Figura 3.1: Circuitos adquisición de voltajes.

Los valores de las resistencias se elegirán de tal manera que la corriente que pase por los divisores de tensión esté alrededor de $0,5\,mA$, para que el consumo de la propia placa sea lo suficientemente pequeño como para poder considerar que no impactará notablemente el consumo del equipo en el que se va a instalar.

Divisor de tensión para +3,3 V

$$\left.\begin{array}{r} \frac{R_2}{R_1+R_2}\cdot 3,3 = 2 \implies R_1 = \frac{1,3}{2}\cdot R_2 \\ R_1 + R_2 = R_T \end{array}\right\} \quad R_T = \frac{3,3}{2}\cdot R_2 \quad \left\{\begin{array}{l} R_1 = \frac{1,3}{3,3}\cdot R_T \\ R_2 = \frac{2}{3,3}\cdot R_T \end{array}\right.$$

$$0,4\,mA < I < 0,6\,mA \implies 5500\,\Omega < R_T < 8250\,\Omega$$

$$\boxed{\begin{array}{l} R_1 = 2210\ \Omega \\ R_2 = 3400\ \Omega \end{array}}$$

Divisor de tensión para +5 V

$$\left.\begin{array}{r} \frac{R_4}{R_3+R_4}\cdot 5 = 2 \Rightarrow R_4 = \frac{2}{3}\cdot R_3 \\ R_3 + R_4 = R_T \end{array}\right\} \quad R_T = \frac{5}{3}\cdot R_3 \quad \left\{\begin{array}{l} R_3 = \frac{3}{5}\cdot R_T \\ R_4 = \frac{2}{5}\cdot R_T \end{array}\right.$$

$$0,4\,mA < I < 0,6\,mA \implies 8334\,\Omega < R_T < 12500\,\Omega$$

$$\boxed{\begin{array}{l} R_3 = 5,36\ k\Omega \\ R_4 = 3,57\ k\Omega \end{array}}$$

Divisor de tensión para +12 V

$$\left.\begin{array}{r} \frac{R_6}{R_5+R_6}\cdot 12 = 2 \Rightarrow R_5 = 5\cdot R_6 \\ R_5 + R_6 = R_T \end{array}\right\} \quad R_T = 6\cdot R_6 \quad \left\{\begin{array}{l} R_5 = \frac{5}{6}\cdot R_T \\ R_6 = \frac{1}{6}\cdot R_T \end{array}\right.$$

$$0,4\,mA < I < 0,6\,mA \implies 20\,k\Omega < R_T < 30\,k\Omega$$

$$\boxed{\begin{array}{l} R_5 = 23,2\ k\Omega \\ R_6 = 4,64\ k\Omega \end{array}}$$

Amplificador para -12 V

$$\frac{v_{in}}{R_7} = -\frac{v_{out}}{R_8} \Rightarrow \left\{ \begin{array}{l} R_7 = -\frac{v_{in}}{v_{out}} \cdot R_8 \\ R_7 = -\frac{-12}{2} \cdot R_8 \end{array} \right\} \Rightarrow R_7 = 6 \cdot R_8$$

$$0,4\,mA \leq I \leq 0,6\,mA \quad \Longrightarrow \quad 20000 \leq R_7 \leq 30000$$

$$\boxed{\begin{array}{l} R_7 = 21,5\ k\Omega \\ R_8 = 3,57\ k\Omega \end{array}}$$

3.1.2. Tratamiento de las señales de los sensores de temperatura

Se va a diseñar este subcircuito para aprovechar sólo la parte de temperaturas positivas del rango del sensor de temperatura, entre $0\,°C$ y $100\,°C$. El sensor LM335 transforma este rango de temperaturas en $(2,73\,V$ a $3,73\,V)$ y se convertirá posteriormente mediante un amplificador operacional en el rango $(0,50\,V$ a $2,50\,V)$. El esquema del subcirtuito diseñado para tal efecto se encuentra en la figura 3.2.

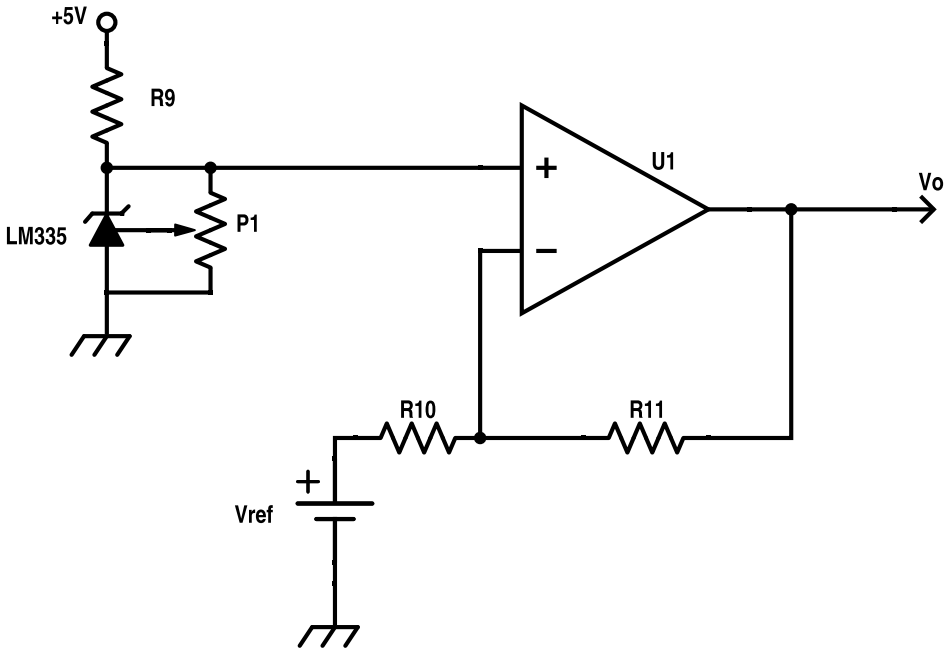

Figura 3.2: Circuito adquisición de temperatura.

Calculamos R_9

La corriente de funcionamiento del sensor LM335 tiene que estar entre $450\,\mu A$ y $5\,mA$

$$450\mu \leq I_R \leq 5m$$
$$450\mu \leq \frac{v_c - v_{LM335}}{R_9} \leq 5m$$
$$450\mu \leq \frac{5-3,73}{R_9} \quad ; \quad \frac{5-2,73}{R_9} \leq 5m$$
$$\frac{5-2,73}{5m} \leq R_9 \leq \frac{5-3,73}{450\mu}$$
$$454\Omega \leq R_9 \leq 2822\Omega$$

Se ha escogido un valor intermedio:

$$\boxed{R_9 = 1780\,\Omega}$$

Calculamos v_{ref}

$$v_o = \left(1 + \frac{R_{11}}{R_{10}}\right) \cdot v_i - v_{ref}$$
$$2,5 = (1+1) \cdot 3,73 - v_{ref}$$
$$v_{ref} = 4,96\,V$$

Para generar v_{ref} se utilizará una fuente de $5\,V$ con un divisor de tensión, quedando el subcircuito completo tal y como se muestra en el esquema de la figura 3.3. Calculamos las resistencias necesarias:

$$v_{ref} = \frac{R_{13}}{R_{12} + R_{13}} \cdot v_{alim}$$
$$5 \cdot R_{13} = 4,96 \cdot (R_{13} + R_{13})$$
$$5 \cdot R_{13} = 4,96 \cdot R_{12} + 4,96 \cdot R_{13}$$
$$0,04 \cdot R_{13} = 4,96 \cdot R_{12}$$
$$R_{13} = 124 \cdot R_{12}$$

Se desea que circuile una corriente mayor que $0,1\,mA$ por lo que se va a elegir unas resistencias para que la corriente a través del divisor sea aproximadamente $5\,mA$:

$$I = \frac{5}{R_{13}} R_{12} + R_{13} \simeq 5\,mA \quad \Longrightarrow \quad R_{12} + R_{13} = 1000\,\Omega$$

El par de resistencias que mejor cumplen estas condiciones son:

$$\boxed{\begin{aligned} R_{12} &= 10\Omega \\ R_{13} &= 1240\Omega \end{aligned}}$$

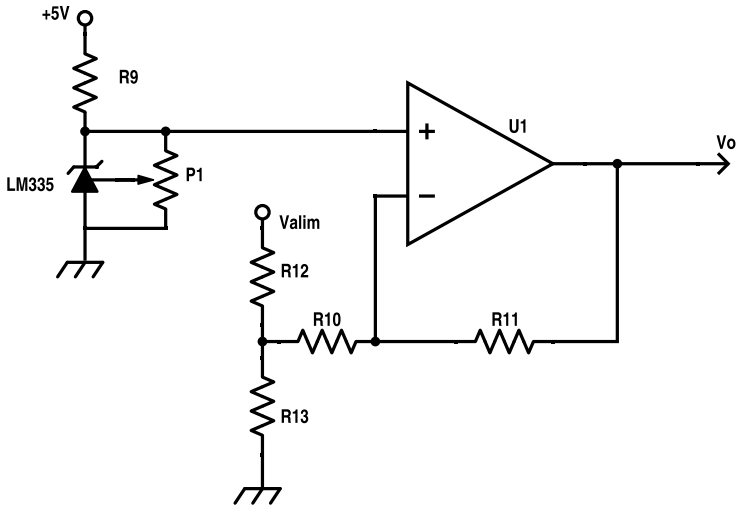

Figura 3.3: Circuito adquisición de temperatura completo.

A continuación se calcula el valor de las resistencias R_{10} y R_{11}

$$v_i = v_+ = v_- \Longrightarrow \frac{v_o - v_i}{R_{11}} = \frac{v_i - v_{ref}}{R_{10}}$$
$$R_{10} \cdot v_0 - R_{10} \cdot v_i = R_{11} \cdot v_i - R_{11} \cdot v_{ref}$$
$$v_0 = \frac{(R_{10} + R_{11}) \cdot v_i - R_{11} \cdot v_{ref}}{R_{10}}$$
$$v_0 = \left(1 + \frac{R_{11}}{R_{10}}\right) \cdot v_i - \frac{R_{11}}{R_{10}} \cdot v_{ref}$$

$$\left.\begin{aligned} T_{(0^oC)}: \quad 0,5 &= \left(1 + \tfrac{R_{11}}{R_{10}}\right) \cdot 2,73 - \tfrac{R_{11}}{R_{10}} \cdot v_{ref} \\ T_{(100^oC)}: \quad 2,5 &= \left(1 + \tfrac{R_{11}}{R_{10}}\right) \cdot 3,73 - \tfrac{R_{11}}{R_{10}} \cdot v_{ref} \end{aligned}\right\} 2 = \left(1 + \frac{R_{11}}{R_{10}}\right) \Longrightarrow R_{10} = R_{11}$$

Se calculan los valores de R_{10} y R_{11} para que la corriente que las atraviese sea de aproximadamente $0,1\,mA$:

$$I_{R_{10}} = I_{R_{11}} \leq 0,1\,mA$$
$$R_{10} \geq \frac{4,96 - 2,73}{0,1\,m} = 22300$$
$$\boxed{R_{10} = R_{11} = 22,6\,k\Omega}$$

3.1.3. Tolerancia de las resistencias

Tal y como se explica en la sección 3.6, los límites de funcionamiento para los voltajes es de un 5 % para las tensiones positivas y de un 10 % para $-12\,V$. Por este motivo, se ha decidido que el error máximo debido a las tolerancias en el caso de los voltajes sea de $0,10\,V$.

Divisor de tensión para 3,3V

$$\text{medida teórica} = \frac{R_2}{R_1+R_2} \cdot v_{a\,medir} = \frac{3400}{2210+3400} \cdot 3,3 = 2{,}00$$

$$\epsilon_{max} > |valor\ obtenido - valor\ ideal|$$

$$\frac{R_2 \cdot (1+T)}{R_1 \cdot (1-T) + R_2 \cdot (1+T)} \cdot v_{a\,medir} - v_{teórica} < \epsilon_{max}$$

$$A = \frac{v_{teórico} + \epsilon_{max}}{v_{a\,medir}} = \frac{2,00 + 0,05}{3,3}$$

$$T < \frac{AR_1 + AR_2 - R_2}{R_2 + AR_1 - AR_2}$$

$$\boxed{T < 6{,}433\,\%}$$

Divisor de tensión para 5V

$$\text{medida teórica} = \frac{R_4}{R_3+R_4} \cdot v_{a\,medir} = \frac{4000}{6000+4000} \cdot 5 = 2,00$$

$$\epsilon_{max} > |valor\ obtenido - valor\ ideal|$$

$$\frac{R_4 \cdot (1+T)}{R_3 \cdot (1-T) + R_4 \cdot (1+T)} \cdot v_{a\,medir} - v_{teorica} < \epsilon_{max}$$

$$A = \frac{v_{teórico} + \epsilon_{max}}{v_{a\,medir}} = \frac{2 + 0,05}{5}$$

$$T < \frac{AR_3 + AR_4 - R_4}{R_4 + AR_3 - AR_4}$$

$$\boxed{T < 4{,}132\,\%}$$

Divisor de tensión para 12V

$$\text{Medida teórica} = \frac{R_6}{R_5+R_6} \cdot v_{a\,medir} = \frac{4000}{20000+4000} \cdot 12 = 2,00$$

$$\epsilon_{max} > |valor\ obtenido - valor\ ideal|$$

Por lo que,

$$\frac{R_6 \cdot (1+T)}{R_5 \cdot (1-T) + R_6 \cdot (1+T)} \cdot v_{a\,medir} - v_{teorica} < \epsilon_{max}$$

$$A = \frac{v_{teórico} + \epsilon_{max}}{v_{a\,medir}} = \frac{2 + 0,05}{12}$$

$$T < \frac{AR_5 + AR_6 - R_6}{R_6 + AR_5 - AR_6}$$

$$\boxed{T < 2{,}941\,\%}$$

Amplificador para -12 V

$$\text{medida teórica} = \frac{-R_8}{R_7} \cdot v_{a\,medir} = \frac{-4000}{24000+4000} \cdot -12 = 2,00$$

$$\epsilon_{max} > |valor\ obtenido - valor\ ideal| = |(G' - G) \cdot v_i|$$

$$\epsilon_{max} > \left| \left(\frac{-R_2 \cdot (1-T)}{R_1(1+T)} + \frac{R_2}{R_1} \right) \cdot v_i \right|$$

$$\epsilon_{max} > \left| \frac{-R_2(1-T) + R_2(1+T)}{R_1(1+T)} \cdot v_i \right|$$

$$\epsilon_{max} > \frac{R_2}{R_1} \cdot \frac{2 \cdot T}{(1+T)} \cdot |v_i|$$

Las fracciones siempre serán positivas, solo v_i puede ser negativa

$$\begin{aligned} \epsilon_{max} \cdot R_1 \cdot (1+T) &> 2 \cdot T \cdot R_2 \cdot |v_i| \\ \epsilon_{max} \cdot R_1 + \epsilon_{max} \cdot R_1 \cdot T &> 2 \cdot T \cdot R_2 \cdot |v_i| \\ \epsilon_{max} \cdot R_1 &> 2 \cdot T \cdot R_2 \cdot |v_i| - \epsilon_{max} \cdot R_1 \cdot T \end{aligned}$$

$$T < \frac{\epsilon_{max} \cdot R_1}{2 \cdot R_2 \cdot |v_i| - \epsilon_{max} \cdot R_1}$$

$$\boxed{T < 2{,}564\,\%}$$

Sensores de temperatura

En este caso, se ha decidido un error máximo por tolerancias inferior a $1\,^oC$.

$$\left.\begin{array}{l} \epsilon = v_o' - v_o \\ T = 50 \cdot v_o - 25 \end{array}\right\} \quad \epsilon_{max} = 1^oC \geq T' - T$$

$$1 \geq 50 \cdot v_o' - 25 - 50 \cdot v_o + 25$$

$$\frac{1}{50} \geq v_o' - v_o$$

Calculamos v'_o:

$$v'_o = \left(1 + \frac{R_{11}\cdot(1+T)}{R_{10}\cdot(1-T)}\right)\cdot v_i - \frac{R_{11}\cdot(1+T)}{R_{10}\cdot(1-T)}\cdot\frac{R_{13}\cdot(1-T)}{R_{12}\cdot(1-T)+R_{13}\cdot(1-T)}\cdot v_{alim}$$

$$v'_o = \left(1 + \frac{1+T}{1-T}\right)\cdot v_i - \frac{(1+T)\cdot R_{13}}{R_{12}\cdot(1-T)+R_{13}\cdot(1-T)}\cdot v_{alim}$$

$$v'_o = \frac{1-T+1+T}{1-T}\cdot v_i - \frac{(1+T)\cdot 124\cdot R_{12}}{R_{12}\cdot(1-T)+124\cdot R_{12}\cdot(1-T)}\cdot v_{alim}$$

$$v'_o = \frac{2}{1-T}\cdot v_i - \frac{124+124\cdot T}{}\cdot v_{alim}$$

$$v'_o = \frac{250\cdot v_i - (124+124\cdot T)\cdot v_{alim}}{125-125\cdot T}$$

$$v'_o = \frac{250\cdot v_i - 620 - 620\cdot T}{125-125\cdot T}$$

Sustituimos v'_o y v_o ($v'_0|_{(T=0)}$) en la equación que calculamos anteriormente:

$$0,02 \geq v'_o - v_o = \frac{250\cdot v_i - 620 - 620\cdot T}{125-125\cdot T} - \frac{250\cdot v_i - 620}{125}$$

$$0,02 + \frac{250\cdot v_i - 620}{125} \geq \frac{250\cdot v_i - 620 - 620\cdot T}{125-125\cdot T}$$

$$125\cdot\left(0,02 + \frac{250\cdot v_i - 620}{125}\right) - 125\cdot\left(0,02 + \frac{250\cdot v_i - 620}{125}\right)\cdot T \leq 250\cdot v_i - 620 - 620\cdot T$$

$$T\cdot\left[620 - 125\cdot\left(0,02 + \frac{250\cdot v_i - 620}{125}\right)\right] \leq 250\cdot v_i - 620 - 125\cdot\left(0,02 + \frac{250\cdot v_i - 620}{125}\right)$$

$$T \leq \frac{250\cdot v_i - 620 - 125\cdot\left(0,02 + \frac{250\cdot v_i - 620}{125}\right)}{620 - 125\cdot\left(0,02 + \frac{250\cdot v_i - 620}{125}\right)}$$

Esto significa que T_{max} se encontrará en uno de los extremos, por lo que calculamos ambos extremos y escogemos la que sea más restrictiva de las dos, que será las más restrictiva de todo el rango.

$$\left.\begin{aligned} T_{v_i=2,73V} &= 0,0045 \\ T_{v_i=3,73V} &= 0,0082 \end{aligned}\right\} \quad \boxed{T \leq 0,45\,\%}$$

El circuito de adquisición de temperaturas es el caso que más limita la elección de la tolerancia de las resistencias en los que $T < 0{,}45\,\%$, y la tolerancia máxima, en catálogo comercial, que cumple este requisito es la de $\boxed{0{,}1\ \%}$.

Como el número de resistencias en el diseño no es muy elevado y vamos a fabricar un único prototipo, todas las resistencias de la parte de adquisición de datos se elegirán con esta tolerancia ya que la diferencia de precio no será significativa.

3.2. Control

Esta etapa contiene el microcontrolador y los componentes necesarios para su correcto funcionamiento y para la comunicación mediantes conexión USB con el ordenador.

Como ya se ha comentado en el estudio previo, para esta sección de la placa de control se ha elegido la placa STM32F3Discovery, una placa con un microcontrolador modelo STM32F303VCT6 que posee una memoria flash de 256 KB, 48 KB de RAM en un empaquetado LQFP100.

Se va a utilizar IAR Embedded Workbench como entorno de desarrollo y programación para la placa [dcc13], y como se ha comentado anteriormente, se programará en C.

La placa recibirá la información necesaria del sistema a través de un único puerto de entrada. Dicho puerto será el correspondiente al conversor analógico digital utilizado (ADC1) que corresponde al pin PC.1. Utilizaremos los registros D y E de la placa como registro de salidas. Se configurará el pin PD.14 como salida del generador de PWM para controlar los ventiladores, y de los nueve pines que están entre el PE.7 y el PE.15, se utilizarán tres (PE.11, PE.13 y PE.15) para controlar qué sensor se lee en cada momento a través del multiplexor analógico, y los otros seis para encender o apagar los LEDs del sistema de alerta.

Pin	Uso
PC.1	Entrada ADC1.
PD.14	Salida PWM para el control de los ventiladores.
PE.7	Salida para el LED del sistema.
PE.8	Salida para el LED de la temperatura.
PE.9	Salida para el LED de $-12\,V$.
PE.10	Salida para el LED de $+12\,V$.
PE.11	Salida para el control del multiplexor analógico (A0).
PE.12	Salida para el LED de $+5\,V$.
PE.13	Salida para el control del multiplexor analógico (A1).
PE.14	Salida para el LED de $+3,3\,V$.
PE.15	Salida para el control del multiplexor analógico (A2).

Tabla 3.1: Pines salida/entrada del microprocesador

3.3. Actuadores

Esta etapa se divide en dos partes: la primera mostrará, mediante LEDs, el estado del sistema, y la segunda se encargará del control de los ventiladores para mantener la temperatura del equipo dentro del rango especificado.

3.3.1. Sistema de alertas

Esta etapa contará con circuitos formados por resistencias y LEDs para crear el sistema de alerta tal y como se muestra en la figura 3.4. Estará formado por los LEDs enumerados como se indica en la tabla 3.2.

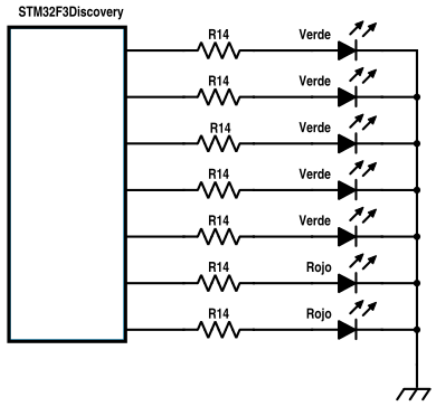

Figura 3.4: Circuito del sistema de alertas.

Valor	Color	Encendido
$+3,3\,V$	Verde	El valor está en rango.
$+5\,V$	Verde	El valor está en rango.
$+12\,V$	Verde	El valor está en rango.
$-12\,V$	Verde	El valor está en rango.
Sistema	Verde	La placa está encendida.
Temperatura	Rojo	Algún sensor de temperatura está fuera de rango.

Tabla 3.2: LEDs del sistema de alerta.

Para que la corriente no sea excesiva a través de los LEDs, se va a colocar una resistencia de $470\,\Omega$ en serie con cada uno de ellos, de esta forma la corriente que circula es aproximadamente:

$$5 - \frac{2}{470} = 6,4\,mA$$

3.3.2. Ventiladores

Para esta etapa se van a utilizar ventiladores estandar de $12\,cm$, con una alimentación de $12\,V$. Independientemente del número de ventiladores, la velocidad de los mismos se controlará mediante una única salida PWM de la placa. Esta señal PWM tiene un nivel alto de unos $3\,V$, por lo que se va a conectar un amplificador operacional TL-081, como los utilizados en la etapa de adquisición de datos, para aumentar el voltaje a $12\,V$ y que sea capaz de suministrar la corriente demandada por los ventiladores.

Se necesita amplificar el rango de salida de la señal PWM ($0\,V$ a $3\,V$) al rango de funcionamiento de los ventiladores ($0\,V$ a $12\,V$), por lo que se va a emplear una configuración no inversora, tal y como se muestra en la figura 3.5, con una ganancia igual a cuatro.

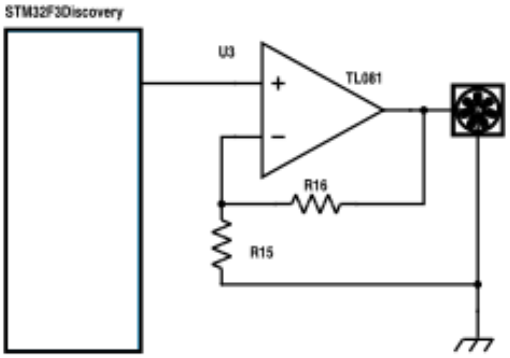

Figura 3.5: Circuito de los ventiladores.

Calculamos R_{15} y R_{16}:

$$v_{ventiladores} = 4 \cdot v_{pwm}$$

$$v_{ventiladores} = \left(1 + \frac{R_{16}}{R_{15}}\right) \cdot v_{pwm}$$

$$\frac{R_{16}}{R_{15}} = 3$$

$$R_{16} = 3 \cdot R_{15}$$

$$\boxed{\begin{aligned} R_{15} &= 1,78\,\Omega \\ R_{16} &= 5,36\,\Omega \end{aligned}}$$

3.4. Esquema completo

En la figura 3.6 se muestra el sistema completo para el caso de emplear dos sensores de temperatura.

En las figuras 3.7 y 3.8 se presenta el rutado"de las capas superior e inferior de la placa respectivamente. Y en las figuras 3.9 y 3.10 se muestran dos fotos de las placas, la primera con todos los valores dentro de rango y la segunda con un error en el valor de $12\,V$.

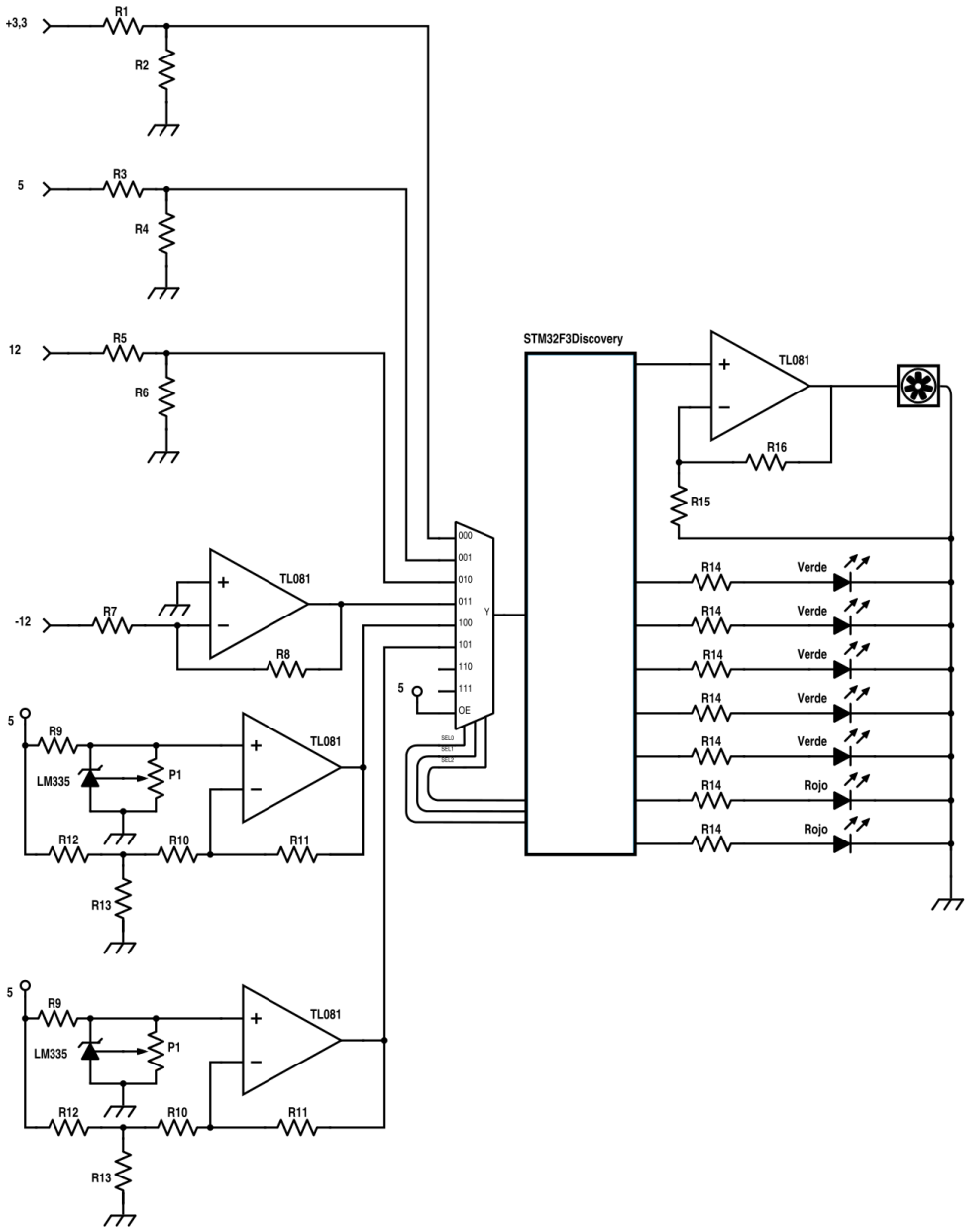

Figura 3.6: Esquema de la placa completa.

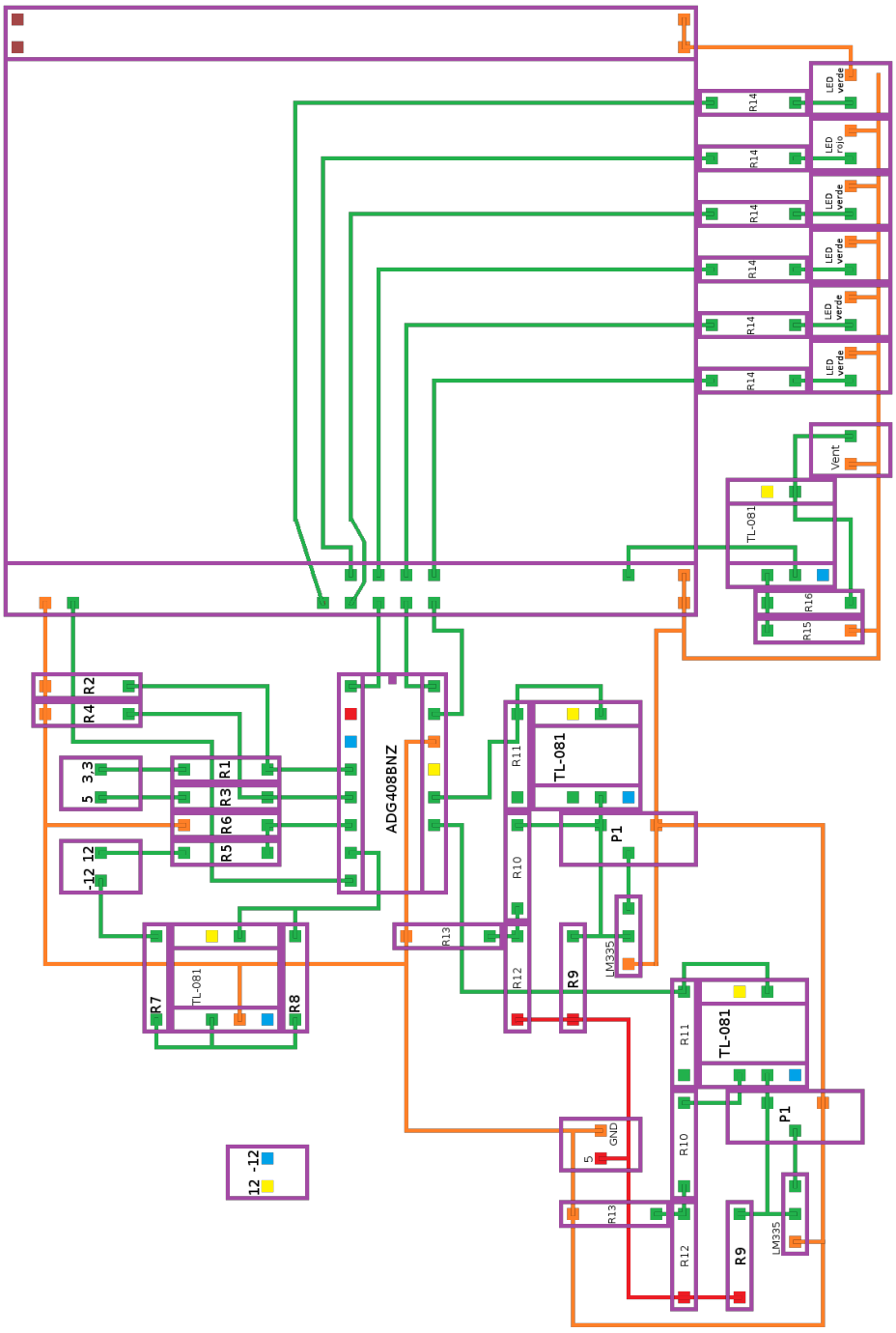

Figura 3.7: Rutado de la cara superior de la placa.

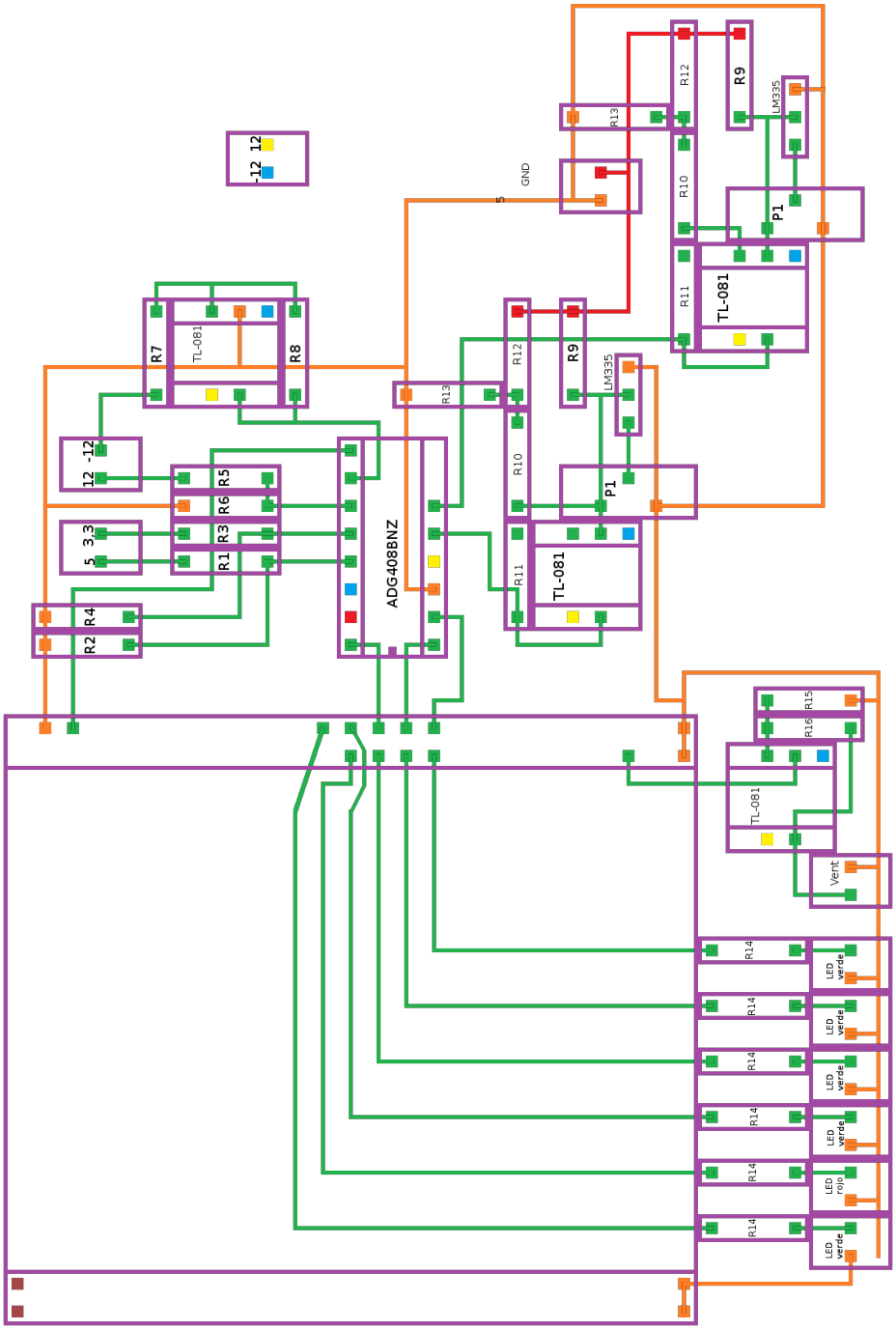

Figura 3.8: Rutado de la cara inferior de la placa.

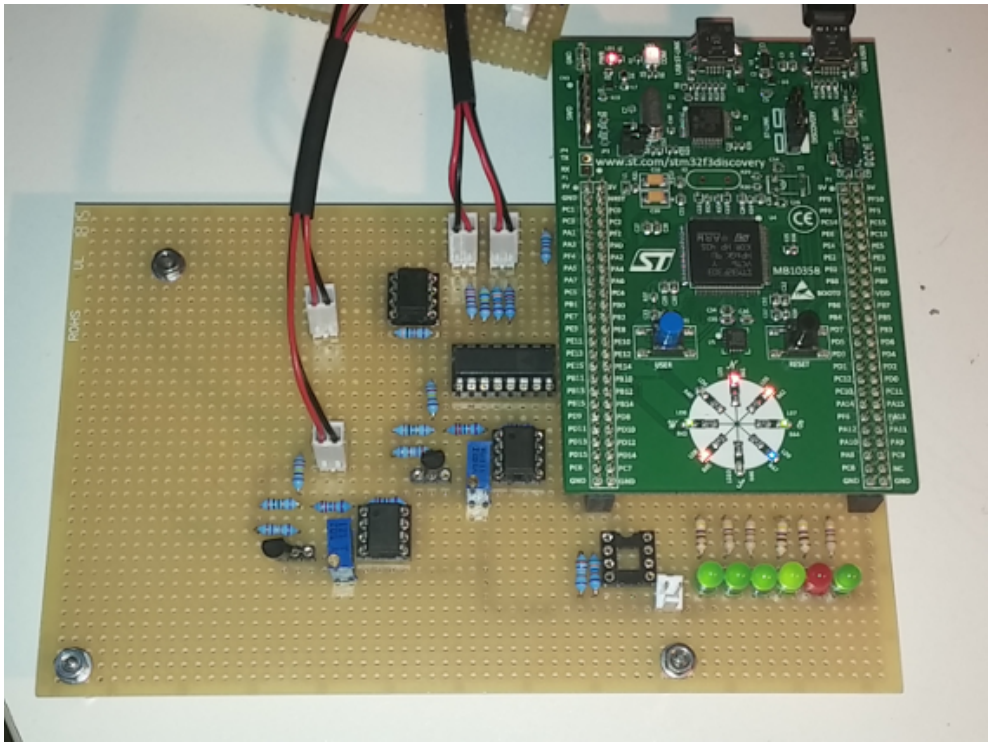

Figura 3.9: Placa con todas las entradas dentro de sus intervalos nominales.

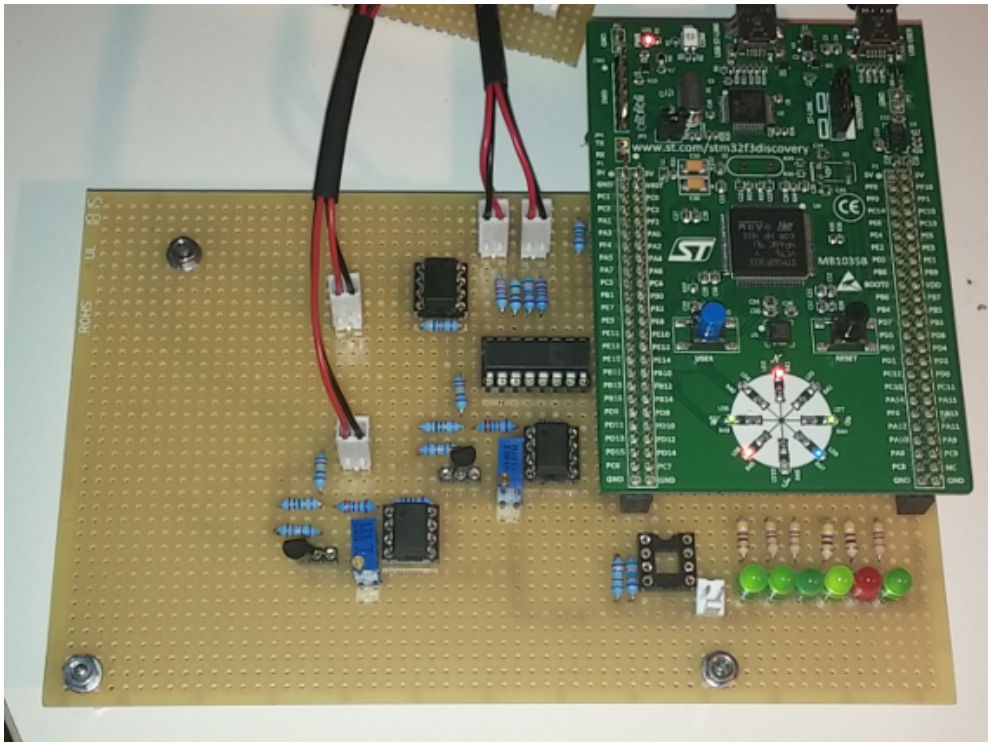

Figura 3.10: Alarma de 12V activada.

3.5. Funciones de transformación

Estas funciones dan la relacción entre los valores reales y los valores que lee el microcontrolador de los convertidores A/D. De esta forma se puede transformar las lecturas tomadas a datos reales y, de forma inversa, calcular los límites de funcionamiento que habrá que poner en la placa de control.

El convertidor A/D de la placa STM32F3Discovery que estamos utilizando tiene una resolución de ocho bits (4096) y lee, en mV, sobre $3\,C$. Por lo que la ecuación que describe la relación entre el valor que le llega al convertidor y lo que recibe el microcontrolador es la siguiente:

$$Val_{micro} = 1000 \cdot \frac{4096}{3000} \cdot v_o$$

3.5.1. Voltajes

En el caso de los voltajes positivos se utiliza un divisor de tensión, por lo que la relación entre el valor leído por el convertidor A/D y el valor real que queremos medir es:

$$v_{leido} = \frac{R_b}{R_a + R_b} \cdot v_{real}$$

Por lo que:

$$Val_{micro} = 1000 \cdot \frac{4096}{3000} \cdot \frac{R_b}{R_a + R_b} \cdot v_{real}$$

Función para +3.3V

$$Val_{micro} = 1000 \cdot \frac{4096}{3000} \cdot \frac{R_2}{R_1 + R_2} \cdot v_{real} = \frac{81920}{99} \cdot v_{real}$$

Función para +5V

$$Val_{micro} = 1000 \cdot \frac{4096}{3000} \cdot \frac{R_4}{R_3 + R_4} \cdot v_{real} = \frac{487424}{893} \cdot v_{real}$$

Función para +12V

$$Val_{micro} = 1000 \cdot \frac{4096}{3000} \cdot \frac{R_6}{R_5 + R_6} \cdot v_{real} = \frac{2048}{9} \cdot v_{real}$$

Función para -12V

En este caso la relación entre el valor leido por el convertidor A/D y el valor real que queremos medir es:

$$v_o = \frac{-R_8}{R_7} \cdot v_{real}$$

$$Val_{micro} = 1000 \cdot \frac{4096}{3000} \cdot \frac{-R_8}{R_7} \cdot v_{real}$$

Por lo que tendríamos la siguiente ecuación:

$$Val_{micro} = 1000 \cdot \frac{4096}{3000} \cdot \frac{-R_8}{R_7} \cdot v_{real} = -\frac{243712}{1075} \cdot v_{real}$$

3.5.2. Temperaturas

En el caso de los sensores de temperatura, el circuito es algo más complejo, pero la relación entre la temperatura medida por el sensor y los voltios que entrega al amplificador operacional es la siguiente:

$$v_i = 10m \cdot (T_{(^oC)} + 273)$$

La siguiente función describe la relación entre la entrada y la salida del amplificador operacional:

$$v_o = 2 \cdot v_i - 4,96$$

Juntando ambas ecuaciones, se obtiene la siguiente función de transformación, que describe el sensor y el amplificador operacional:

$$V_o = 20m \cdot T_{(^oC)} + 0,5$$

Y añadiéndola a la ecuación del principio. Se obtiene la función que describe el comportamiento de toda la etapa y que nos da la relación entre el valor que recibe el micro y la temperatura que mide el sensor:

$$Val_{micro} = 1000 \cdot \frac{4096}{3000} \cdot \left(20m \cdot T_{(^oC)} + 0,5\right)$$

$$Val_{micro} = \frac{2048}{75} \cdot T_{(^oC)} + \frac{2048}{3}$$

3.6. Límites de funcionamiento

3.6.1. Voltaje

La tolerancia normal en los niveles de voltaje de una fuente de alimentación comercial es de entre un 5 % ó 10 %. En la tabla 3.3 se encuentran detallados los valores mínimos y máximos reales de cada voltaje, así, como su valor en el microcontrolador.

Medida	Tolerancia	Mínimo		Máximo	
3,3 V	5 %	3,135 V	2594	3,465 V	2867
5 V	5 %	4,75 V	2593	5,25 V	2866
12 V	5 %	11,4 V	2594	12,6 V	2867
-12 V	10 %	10,8 V	2448	13,2 V	2993

Tabla 3.3: Valores máximos y mínimos de voltaje.

3.6.2. Temperatura

La temperatura máxima de funcionamiento variará dependiendo del equipo, por lo que los valores de esta sección habría que modificarlos para adaptarlos al equipo que fuese a controlar la placa. Una temperatura de trabajo máxima recomendada para estos equipos suele ser $55\,^0C$, por lo que se utilizará este valor para el prototipo.

Para calcular las franjas de funcionamiento de los ventiladores, se toma como mínimo hipotético $25\,^0C$, y haremos que trabajen a cuatro potencias diferentes (25 %, 50 %, 75 % y 100 %,). Las potencias mínima y máxima serán aplicadas a los valores por debajo y por encima de los límites nominales, por lo que su intervalo de funcionamiento dentro de los límites será la mitad que el de las dos potencias intermedias. Las potencias de funcionamiento y sus limites se encuentran en la tabla 3.4.

Potencia	Mínimo		Máximo	
25 %	—	—	30 ^{0}C	1502
50 %	30 ^{0}C	1502	40 ^{0}C	1775
75 %	40 ^{0}C	1775	50 ^{0}C	2048
100 %	50 ^{0}C	2048	—	—

Tabla 3.4: Temperaturas de funcionamiento de los ventiladores.

El LED de alerta se activará cuando la placa detecte una temperatura superior a $55\,^oC$.

Capítulo 4

Programación

4.1. Comunicación

La comunicación entre la placa de control y el ordenador se realizará mediante un cable USB-macho/microUSB-macho estandar.

La placa enviará periódicamente la velocidad de los ventiladores, el nivel de las cuatro tensiones que es capaz de medir ($3\,V$, $5\,V$, $12\,V$ y $-12\,V$) y los valores obtenidos de los sensores de temperatura. El microcontrolador guardará todos estos valores, separados por guiones, en una cadena que luego enviará a través del USB haya o no un ordenador conectado al otro extremo.

A continuación se encuentra la cadena que se enviaría si los voltajes coincidiesen exactamente con sus valores nominales y los sensores de temperatura marcasen 35 y 45 grados cada uno:

$$37 - 2730 - 2729 - 2731 - 2721 - 1638 - 1911 - \backslash n$$

$$(Ventiladores) - (3,3\,V) - (5\,V) - (12\,V) - (-12\,V) - (Temp1) - (Temp2) - \backslash n$$

Se ha añadido un guión al final, para facilitar la extración de cada valor, y un

salto de línea para que, si fuera necesario conectarse a la placa y ver su salida sin utilizar el programa diseñado para ello, sea más sencillo.

En el ordenador, el programa leerá del puerto serie hasta que le llegue un set de datos, los procesará y volverá a leer hasta que le llegue el siguiente set.

Los siguientes puntos constituyen algunas características generales que hay que tener en cuenta en este protocolo de comunicaciones:

- Las respuestas siempre van a tener una longitud fija de 36 bytes.
- Los datos para el voltaje -12V se comunicarán en valor absoluto, ya que la placa trabaja con ellos en esta forma.
- Los datos serán tratados en dos formatos: el valor real que utilizará el ordenador y que será el que muestra la interfaz; y uno relativo, que será el que leerá el microcontrolador de los sensores y que será el único que vea el microcontrolador, este valor estará comprendido entre 0 y 3000. Todas las transmisiones serán en estos valores relativos. El programa de ordenador será el encargado de hacer las transformaciones entre formatos.
- El puerto USB se ha configurado a 9600 baudios y una paridad de 8N1.

4.2. Programas

El código que se ha escrito para el microcontrolador está contenido en un único archivo denominado *main.c*. Además, se utilizan multiples drivers que se pueden descargar gratuitamente desde la página de STMicroelectronics. El código para el microcontrolador se encuentra en la página 84. Este código se encarga de configurar el multiplexor.

El programa está escrito utilizando C como lenguaje principal. Para la interfaz gráfica se ha utilizado Gtk+ versión 3.4, mediante la herramienta gráfica Glade (versión 3.12.1). El programa se divide en dos procesos mediante un *fork*. El proceso padre será el encargado de gestionar el interfaz gráfico y sus peticiones, mientras que el proceso hijo se encargará de la comunicación con la placa y de realizar el registro de datos.

El intercambio de información entre procesos se realizará con la información en el formato utilizado por el micro y será el proceso padre el encargado de transformarla

a valores reales, salvo cuando se active el registro de datos que lo hará también el proceso hijo para guardar la información en dicho formato.

4.3. Archivos del Programa de Ordenador

child.h Este archivo contiene el código del proceso hijo. Este código se encarga de la comuniación con la placa, transformar los datos a valores reales y guardar los datos a un fichero de log. El código está en la página 65.

functions.h Este archivo contiene el código de las funciones de todo el programa excepto de la función main. El código está en la página 69.

logo.jpg Logo que se muestra en el programa.

gui.glade Este archivo contiene la salida que genera Glade, es XML pero no se puede utilizar directamente. Es necesario modificar un par de etiquetas:

- Sustituir *interface* por *glade-interface*.
- Procesarlo con gtk-builder-convert.

gui.xml Este archivo contiene el XML ya procesado y es el que se cargará desde *parent.h*.

headers.h Este archivo contiene las cabeceras de las funciones con una explicación de lo que hace la función. El código está en la página 70.

main.c Este archivo contiene los includes, la inicialización de la memoria compartida y el *fork* que crea el proceso padre y el proceso hijo. El código está en la página 73.

parent.h Este archivo contiene el código del proceso padre. Este código se encarga de la ventana de la herramienta, tanto de refrescar la información como de manejar la señales provenientes de la misma. El código está en la página 74.

programa Este archivo es el resultado de la compilación y es el que habrá que ejecutar para arrancar el programa.

structures.h Este archivo contiene la descripción de la estructura utilizada para almacenar los datos. El código está en la página 76.

4.4. Funciones del Programa de Ordenador

void check_failure(char *message, float data, float min, float max)

Esta función se usa para comprobar si un valor está dentro de su rango de trabajo, devuelve una variable de tipo char con cinco espacios si el valor está dentro de su rango o "FAILURE"si no es así.

on_window_destroy(GtkButton *object, Data_Structure *data)

Esta función se ejecuta cuando la ventana del programa se cierra. Se encarga de liberar la memoria compartida y de enviarle la orden de finalizar al proceso hijo.

parse_data(Data_Structure *data)

Esta función divide la cadena recibida desde el microcontrolador utilizando los guiones como separadores y convierte cada subcadena en datos de tipo *float* para que el programa pueda realizar comparaciones con ellos.

recording(GtkButton *object, Data_Structure *data)

Esta función se ejecuta cuando se hace click en el *Togglebutton tb_rec* de la interfaz. Pone el proceso hijo en modo registro cuando el botón está activado, y en modo espera cuando no lo está.

refresh_window(GtkButton *object, Data_Structure *data)

Esta función es un bucle que actualiza la interfaz de usuario con los últimos datos leídos del microcontrolador.

stop_refresh_window(GtkButton *object, Data_Structure *data)

Esta función cambia el valor de la variable que hace que se ejecute el bucle de la función refresh_window para que se deje de refrescar la información de la interfaz de usuario. Si se estuviese guardando la información a un fichero de log, se seguiría guardando.

transform_data(Data_Structure *data)

Esta función transforma los datos para pasarlos de datos del microcontrolador a datos reales de voltajes y temperaturas.

update_data(Data_Structure *data)

Esta función actualiza la información que muestra la interfaz con los valores contenidos en la estructura de tipo Data_Structure del proceso padre.

update_field_text(char* message, GtkWidget *field)

Esta función actualiza el contenido del campo de texto del interfaz que le pasan como argumento con la cadena de caracteres que le pasan.

4.5. Captura de pantalla

Captura de pantalla del programa de ordenador con un error debido a que la lectura de $12\,V$ está por debajo del límite de funcionamiento nominal.

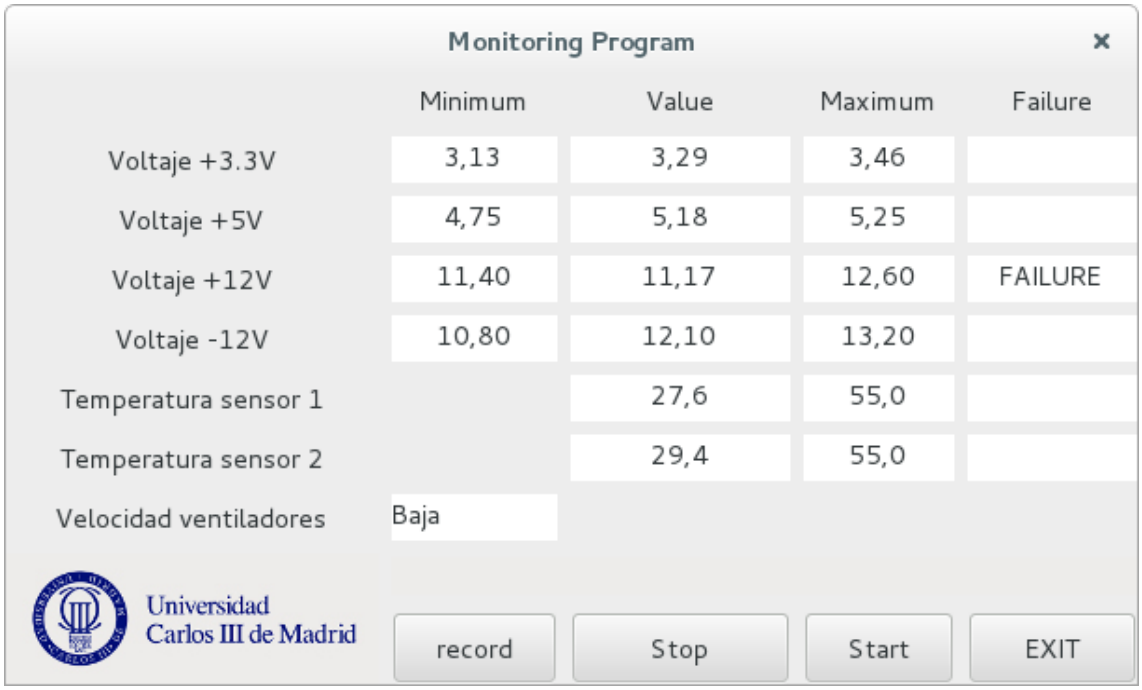

Figura 4.1: Captura del programa de ordenador.

Capítulo 5

Conclusiones y Trabajos Futuros

5.1. Conclusiones

En este proyecto se ha diseñado un sistema capaz de monitorizar las tensiones proporcionadas por la fuente de tensión y la temperatura en un chasis VME y, dependiendo de los datos obtenidos en cada momento, ajustar la velocidad de unos ventiladores e informar al usuario mediante unos LEDs en el propio equipo, si alguno de estos valores se encontraba fuera de sus rangos de funcionamiento nominal. También se ha desarrollado un interfaz de comunicaciones para poder monitorizar estos datos desde un ordenador, para poder ver los valores precisos en cada momento, guardar esta información y poder revisarla a posteriori.

A tal efecto, se ha diseñado una placa hardware, que se encarga de toda la primera parte y un programa para ordenador, que se encargará de mostrar la información y almacenarla en un log si fuese preciso.

La placa es capaz de detectar, tanto pequeñas variaciones de voltajes, dentro y fuera de los rangos nominales, así como caídas totales de las tensiones proporcionadas por la inmensa mayoría de las fuentes comerciales para equipos electrónicos ($3,3\,V$, $5\,V$, $12\,V$ y $-12\,V$). En cuanto a temperaturas, el prototipo creado cuenta con dos

sensores y se le podrían añadir otros dos, detectando temperaturas entre 0 y 100 grados. Este rango engloba los límites de funcionamiento de practicamente cualquier equipo electrónico con el que se pueda trabajar hoy en día.

El programa de ordenador escrito es capaz de comunicarse con la placa, mostrar los datos que va obteniendo de la placa mediante una interfaz visual y guardarlos en un fichero de log si así se le indica.

Debido a lo expuesto anteriormente, se han cumplido los objetivos de este proyecto, ya que aunque la solución creada no cubre la totalidad de los equipos electrónicos que hay en el mercado, sí es capaz de monitorizar y controlar un porcentaje muy elevado de ellos. En el caso de los chasis VME, que utilizan los valores de voltaje anteriormente citados, se podría monitorizar sin modificar este sistema cualquier chasis que no requiriese más de cuatro sensores de temperatura.

5.2. Mejoras y trabajos futuros

Se han detectado diferentes puntos donde se podría mejorar el sistema actual para hacerlo más completo:

- El primer punto está orientado a mejorar las medidas obtenidas por el sistema. Actualmente no son excesivamente precisas debido a que se basan en los niveles de su alimentación que no son excesivamente precisos. Esto además genera algo más de imprecisión en el convertidor analógico digital. Incrementando la calidad de los niveles de alimentación se mejorarían ambos aspectos.

- También se podría mejorar el sistema en cuanto a su robustez diseñando una placa a medida con un microcontrolador y un convertidor analógico/digital que sustituyese a la STM32F3Discovery. Esto también podría reducir el consumo de todo el conjunto.

- Para este proyecto, se han considerado valores de desviación estándar en las fuentes para estimar los rangos de funcionamiento correcto de éstas. Este puede no ser siempre el caso, por lo que también sería interesante adaptar los programas para poder modificar los niveles de funcionamiento desde el ordenador. Ahora se puede hacer, pero es necesario modificar el código de la herramienta de monitorización, modificar el programa de la placa y reprogramarla, lo que lo hace muy pesado si con la misma herramienta se quiere monitorizar equipos con diferentes niveles.

- Este proyecto ha creado una herramienta de monitorización para Linux, lo que hace que necesitemos un ordenador con este sistema operativo instalado. Esto no siempre es cómodo o factible, por lo que sería una buena mejora migrar esta herramienta para hacerla multiplataforma.

- Estos equipos no suelen estar delante de los operadores encargados de ellos, por lo que estaría bien añadir la posibilidad de conexión por Ethernet en la placa para poder monitorizar el equipo a distancia sin necesidad de estar delante o conectarle un equipo físico externo.

Bibliografía

[dcc13] dccharacter.blogspot.com.es. Howto create a new project for stm32f3-discovery in iar from scratch (step-by-step). http://dccharacter.blogspot.com.es/2013/02/creating-new-project-for-stm32f3.html, 2013. [Internet; consultado 21-noviembre-2013].

[GC02a] Félix García Carballeira. *El lenguaje de programación C: diseño e implementación de programas*. Prentice Hall, 2002.

[GC02b] Félix García Carballeira. *Problemas resueltos de programación en lenguaje C*. Thomson Paraninfo, 2002.

[IEE02] IEEE. Vme ieee standard 1014-1987. http://ieeexplore.ieee.org/servlet/opac?punumber=2597, 2002. [Internet; consultado 12-enero-2012].

[Kra07] Andrew Krause. *Foundations of GTK+ development*. Apress, 2007.

[VV.] VV.AA. Gnome developer. https://developer.gnome.com/gtk3/stable/. [Internet; consultado 2014/2015].

[VV.09] VV.AA. Introduction to vme / vxi / vxs standards. http://file.wiener-d.com/documentation/General/WIENER-VME-VXI-VXS-introduction-1.0.pdf, 2009. [Internet; consultado 18-marzo-2014].

[Wik12a] Wikipedia. Vme. https://es.wikipedia.org/wiki/VME, 2012. [Internet; consultado 11-enero-2012].

[Wik12b] Wikipedia. Vmebus. https://en.wikipedia.org/wiki/VMEbus, 2012. [Internet; consultado 11-enero-2012].

Apéndice A

Presupuesto

A continuación se presentan los costes de realización de este Proyecto Fin de Carrera. Para calcularlos se han tenido en cuenta los costes de mano de obra y de materiales.

En la figura A.1 se muestran las fases de desarrollo del proyecto con el número de horas invertidas en realizar cada una de ellas. También se muestra su coste asociado utilizando una tarifa de 15 €/hora. En la figura A.2 se pueden ver los componentes utilizados para fabricar un prototipo de la placa y sus precios, aunque estos valores podrían verse reducidos notablemente si se fabrican más unidades.

El coste de la mano de obra es de 4080,00 €, y el de los materiales 57,39 €, por lo que el coste total del proyecto asciende a 4137,39 €.

Concepto	Horas	Subtotal
Estudio previo	40	600 €
Diseño	80	1200 €
Componentes (Selección y adquisición)	32	480 €
Fabricación	8	120 €
Programación	80	1200 €
Integración	16	240 €
Pruebas	16	240 €
TOTAL	**272**	**4080 €**

Tabla A.1: Coste del diseño

Concepto	Cantidad	p/u	Subtotal
STM32f3Discovery	1	11,39 €	11,39 €
Resistencias	26	0,02 €	0,52 €
Multiplexor Analogico	1	5,71 €	5,71 €
Potenciometros	2	1,14 €	2,28 €
Sensor LM335	2	1,20 €	2,40 €
A.O. (TL081)	4	0,47 €	1,88 €
Ventilador	1	11,00 €	11,00 €
LEDs	6	0,10 €	0,60 €
Conector macho para PCB 50 pines/2 filas	2	2,70 €	5,40 €
Placa de matriz de soldadura	1	6,21 €	6,21 €
Varios (cable, estaño, conectores,...)	1	10,00 €	10,00 €
TOTAL			**57,39 €**

Tabla A.2: Coste de los componentes de la placa.

Apéndice B

Componentes

En este apéndice encontramos una tabla con todos los componentes del proyecto, las tablas con los pines de los chips utilizados y a que van conectados y enlaces a las hojas de características de los componentes que lo requieren.

B.1. Listado completo

Todas las resistencias que se van a utilizar son de la Serie R de TE Connectivity. Esta serie tiene una tolerancia de $0,1\,\%$, tal y como se calculó anteriormente en el apartado 3.1.3 (página 27).

Nombre	Valor	Description
STM32F3		Placa de control (STM32F3Discovery).
TL081IP		Amplificador operacional adquisición $-12\,V$, temperaturas y control de los ventiladores (x4).
LM335		Sensor de temperatura.
LED1	Rojo	Sistema de alertas (x6).
LED2	Verde	Sistema de alertas (x1).
Zócalo		Zocalo 50 pines 2 filas para conectar la STM32F3Discovery a la placa (x2).
Ventilador		
R_1	$2,21\,k\Omega$	Acondicionamiento de la entrada de $+3,3\,V$.
R_2	$3,40\,k\Omega$	Acondicionamiento de la entrada de $+3,3\,V$.
R_3	$5,36\,k\Omega$	Acondicionamiento de la entrada de $+5\,V$.
R_4	$3,57\,k\Omega$	Acondicionamiento de la entrada de $+5\,V$.
R_5	$23,2\,k\Omega$	Acondicionamiento de la entrada de $+12\,V$.
R_6	$4,64\,k\Omega$	Acondicionamiento de la entrada de $+12\,V$.
R_7	$21,50\,k\Omega$	Acondicionamiento de la $-12\,V$.
R_8	$3,570\,k\Omega$	Acondicionamiento de la $-12\,V$.
R_9	$1,78\,k\Omega$	Acondicionamiento de los sensores de temperatura.
R_{10}	$22,60\,k\Omega$	Acondicionamientos de los sensores de temperatura.
R_{11}	$22,60\,k\Omega$	Acondicionamientos de los sensores de temperatura.
R_{12}	$10,00\,\Omega$	Acondicionamientos de los sensores de temperatura.
R_{13}	$1,24\,k\Omega$	Acondicionamientos de los sensores de temperatura.
R_{14}	$470\,\Omega$	Resistencias de los LEDs (x8).
R_{15}	$1,78\,k\Omega$	Resistencia amplificador de los ventiladores.
R_{16}	$5,36\,k\Omega$	Resistencia amplificador de los ventiladores.
P_1	$5\,k\Omega$	Potenciómetro de calibración de los LM335 (x2).

Tabla B.1: Tabla de componentes.

B.2. Componentes

B.2.1. Amplificador Operacional

Se ha elegido el modelo TL-081IP de Texas Instruments de encapsulado PDIP y ocho pines para montaje en orificio pasante. El TL-081 es un amplificador muy común, por lo que está muy probado y es seguro.

Conteo de Pines	8
Dimensiones	9.81 x 6.35 x 4.57 mm
Ganancia de Tensión Típica	106,02 dB
Número de Canales por Chip	1
Producto de Ancho de Banda de Ganancia Típica	3MHZ
Slew Rate Típico	13 $\frac{V}{\mu s}$
Temperatura de Funcionamiento Mínima	$-40\,^oC$
Temperatura de Funcionamiento Máxima	$+85\,C$
Tensión de Alimentación Dual Típica	$\pm 12\,V$, $\pm 15\,V$, $\pm 5\,V$, $\pm 9\,V$

Tabla B.2: Características del amplificador operacional.

Entrada	Pin	Pin	Entrada
Offset N1	1	8	No internal connection
In-	2	7	V_{cc+}
In+	3	6	OUT
V_{cc-}	4	5	Offset N2

Tabla B.3: Pines TL-081.

B.2.2. Multiplexor analógico

Se ha elegido el modelo ADG408BNZ de Analog Devices de encapsulado PDIP y dieciseis pines para montaje en orificio pasante. Se ha elegido este modelo porque se trata de un multiplexor 8x1, que cubre las necesidades básicas de cualquier sistema, cuatro entradas para voltajes y cuatro entradas para sensores de temperatura de las cuales se utilizarán solo dos en la placa prototipo.

Arquitectura del Multiplexor	8:1
Dimensiones	20.07 x 6.35 x 3.3mm
Número de Canales por Chip	1
Temperatura de Funcionamiento Mínima	$-40\,^oC$
Temperatura Máxima de Funcionamiento	$+85\,^oC$
Tensión de Alimentación Dual Típica	$\pm 15\,V$
Tensión de Alimentación Única Típica	12 V

Tabla B.4: Caracterísitcas del multiplexor analógico.

A2	A1	A0	EN	Entrada
X	X	X	0	NONE
0	0	0	1	S_1
0	0	1	1	S_2
0	1	0	1	S_3
0	1	1	1	S_4
1	0	0	1	S_5
1	0	1	1	S_6
1	1	0	1	S_7
1	1	1	1	S_8

Tabla B.5: Tabla de verdad del multiplexor analógico.

Registro	Pin	Pin	Registro
A0	1	16	A1
EN	2	15	A2
V_{ss}	3	14	GND
S1	4	13	V_{DD}
S2	5	12	S5
S3	6	11	S6
S4	7	10	S7
D	8	9	S8

Tabla B.6: Pines del multiplexor analógico.

B.2.3. Sensor de temperatura

Hemos elegido un sensor LM335 modelo LM335AZ fabricado por STMicroelectronics de encapsulado TO-92.

Alto	5.33 mm
Ancho	4.2 mm
Número de pines	3
Ganancia del sensor	10 $\frac{mV}{C}$
Temperatura mínima de funcionamiento	-40 C
Temperatura máxima de funcionamiento	+125 C
Tensión mínima de alimentación	5 V
Tensión máxima de alimentación	40 V
Tipo de encapsulado	TO-92
Tipo de montaje	Montaje en orificio pasante

Tabla B.7: Características del sensor de temperatura LM335.

B.2.4. STM32F3Discovery

Esta placa de desarrollo de la serie de microprocesadores STM32F3 monta un microcontrolador STM32F303VCT6, una flash de 256 KB y 48 KB de RAM. Cuenta con un USB para programarla, otro para comunicar la placa con otros equipos y cien pines de entrada/salida. Se puede alimentar a través del USB o desde fuentes externas de 3 V o 5 V.

Registro	Pin	Pin	Registro
3V	1	26	3V
GND	2	27	NRST
PC.1	3	28	PC.0
PC.3	4	29	PC.2
PA.1	5	30	PF.2
PA.3	6	31	PA.0
PF.4	7	32	PA.2
PA.5	8	33	PA.4
PA.7	9	34	PA.6
PC.5	10	35	PC.4
PB.1	11	36	PB.0
PE.7	12	37	PB.2
PE.9	13	38	PE.8
PE.11	14	39	PE.10
PE.13	15	40	PE.12
PE.15	16	41	PE.14
PB.11	17	42	PB.10
PB.13	18	43	PB.12
PB.15	19	44	PB.14
PD.9	20	45	PD.8
PD.11	21	46	PD.10
PD.13	22	47	PD.12
PD.15	23	48	PD.14
PC.6	24	49	PC.7
GND	25	50	GND

Registro	Pin	Pin	Registro
5V	51	76	5V
PF.9	52	77	PF.10
PF.0	53	78	PF.1
PC.14	54	79	PC.15
PE.6	55	80	PC.13
PE.4	56	81	PE.5
PE.2	57	82	PE.3
PE.0	58	83	PE.1
PB.8	59	84	PB.9
BOOT0	60	85	VDD
PB.6	61	86	PB.7
PB.4	62	87	PB.5
PD.7	63	88	PB.3
PD.5	64	89	PD.6
PD.3	65	90	PD.4
PD.1	66	91	PD.2
PC.12	67	92	PD.0
PC.10	68	93	PC.11
PA.14	69	94	PA.15
PF.6	70	95	PA.13
PA.12	71	96	PA.11
PA.10	72	97	PA.9
PA.8	73	98	PC.9
PC.8	74	99	NIC
GND	75	100	GND

Tabla B.8: Pines de la placa STM32F3Discovery.

B.3. Hojas de características

A contiuación están los enlaces a las hojas de características de todos los componentes utilizados en la placa de control.

Amplificador operacional TL081

http://docs-europe.electrocomponents.com/webdocs/0623/0900766b8062356f.pdf

LEDs

http://docs-europe.electrocomponents.com/webdocs/0026/0900766b80026dcc.pdf

Multiplexor analógico

http://docs-europe.electrocomponents.com/webdocs/077f/0900766b8077ff43.pdf

Placa matriz

http://docs-europe.electrocomponents.com/webdocs/078d/0900766b8078d7eb.pdf

Placa STM32F3Discovery

http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF254044

Potenciómetros

http://docs-europe.electrocomponents.com/webdocs/069c/0900766b8069ccf4.pdf

Resistencias

http://docs-europe.electrocomponents.com/webdocs/1070/0900766b810706c2.pdf

Sensor LM335

http://docs-europe.electrocomponents.com/webdocs/135f/0900766b8135fb7b.pdf

Apéndice C

Código fuente

C.1. Programa para el ordenador

A continuación se encuentra todo el código fuente del programa para el ordenador.

child.h

```
// Open log file in case the recording mode is enable.
FILE *file_log;
file_log=fopen("test.txt", "w");

// Configure Serial Port.
int fd_serie;  // File descriptor
int num;
fd_serie = open_port();

// Read the configureation of the port
struct termios options;
tcgetattr( fd_serie, &options );

// SEt Baud Rate
cfsetispeed( &options, B9600 );
```

```
cfsetospeed( &options, B9600 );
options.c_cflag |= ( CLOCAL | CREAD );

// Set the Charactor size
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS8;

// Set parity - No Parity (8N1)
options.c_cflag &= ~PARENB;
options.c_cflag &= ~CSTOPB;
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS8;

// Disable Hardware flowcontrol
// Enable Raw Input
options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
// Disable Software Flow control
options.c_iflag &= ~(IXON | IXOFF | IXANY);
// Chose raw (not processed) output
options.c_oflag &= ~OPOST;
if ( tcsetattr( fd_serie, TCSANOW, &options ) == -1 )
  printf ("Error with tcsetattr = %s\n", strerror ( errno ) );
fcntl(fd_serie, F_SETFL, FNDELAY);

while(*data->order > -50) {
  // Leemos del puerto serie hasta que la placa manda un set de datos.
  num = -1;
  while ( num < 0 )
    num = read( fd_serie, data->req_data, 36 );
  printf("datos: %s\n", data->req_data);

  parse_data(data);
  //Record
  if ( *data->order == 1 )
    fprintf(file_log,"%.2f-%.2f-%.2f-%.2f-%.2f-%.2f-%.0f\n",  *data->v3,
                      *data->v5,
                      *data->v12,
                      *data->vm12,
                      *data->t1,
                      *data->t2,
                      *data->fan_speed);
}

// Closing log file
fclose(file_log);
```

```
// Closing serial port
close( fd_serie );
```

functions.h

```
void check_failure(char *message, float data, float min, float max) {
        if ((data<=max) && (data>=min))
                sprintf(message,"   ");
        else
                sprintf(message,"FAILURE");
}

void on_window_destroy(GtkButton *object, Data_Structure *data) {
        int status;

  *data->refreshing = 0;

        *data->order = -100;

        waitpid(data->pid, &status, 0);  // BORRAR ANTES DE ENTREGAR.

        shmdt(data->order);
        shmdt(data->req_data);
  shmdt(data->refreshing);
  shmdt(data->v3);
  shmdt(data->v5);
  shmdt(data->v12);
  shmdt(data->vm12);
  shmdt(data->t1);
  shmdt(data->t2);
  shmdt(data->fan_speed);

        // dealocate the shared memory segment.
        shmctl(data->seg_order, IPC_RMID, 0);
        shmctl(data->seg_req_data, IPC_RMID, 0);
  shmctl(data->seg_refreshing, IPC_RMID, 0);
  shmctl(data->seg_v3, IPC_RMID, 0);
  shmctl(data->seg_v5, IPC_RMID, 0);
  shmctl(data->seg_v12, IPC_RMID, 0);
  shmctl(data->seg_vm12, IPC_RMID, 0);
  shmctl(data->seg_t1, IPC_RMID, 0);
  shmctl(data->seg_t2, IPC_RMID, 0);
  shmctl(data->seg_fan_speed, IPC_RMID, 0);

        gtk_main_quit();
}
```

```
void recording(GtkButton *object, Data_Structure *data) {
    if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(data->b_rec)))
                *data->order = 1;
    else
                *data->order = 0;
}

void stop_refresh_window(GtkButton *object, Data_Structure *data) {
  *data->refreshing = 0;
}

void transform_data(Data_Structure *data) {
  // Tratamos los datos.
  *data->v3 = (float)*data->v3 * 99 / 81920;
  *data->v5 = *data->v5 * 893 / 487424;
  *data->v12 = *data->v12 * 9 / 2048;
  *data->vm12 = *data->vm12 * 1075 / 243712;
  *data->t1 = (*data->t1 -3/2048) * 75 / 2048;
  *data->t2 = (*data->t2 -3/2048) * 75 / 2048;
}
void parse_data(Data_Structure *data) {
  char *token;
  token = strtok(data->req_data , "-");
  *data->fan_speed = strtod(token,NULL);
  token = strtok(NULL,"-");
  *data->v3 = strtod(token,NULL);
  token = strtok(NULL,"-");
  *data->v5 = atof(token);
  token = strtok(NULL,"-");
  *data->v12 = atof(token);
  token = strtok(NULL,"-");
  *data->vm12 = atof(token);
  token = strtok(NULL,"-");
  *data->t1 = atof(token);
  token = strtok(NULL,"-");
  *data->t2 = atof(token);

  transform_data(data);
}

void refresh_window(GtkButton *object, Data_Structure *data) {
  if (*data->refreshing == 1) {
    *data->refreshing = 0;
  } else {
    *data->refreshing = 1;
```

```
    while (*data->refreshing == 1) {
      update_data(data);
      while (gtk_events_pending ()) gtk_main_iteration();
      sleep(1);
    }
  }
}

void update_data(Data_Structure *data) {
    char message[99];

    // write 3V data
    sprintf(message,"%.2f", *data->v3);
    update_field_text(message, data->t_3v);
    // write 3V min
    sprintf(message,"%.2f", data->v3_min);
    update_field_text(message, data->t_3v_min);
    // write 3V max
    sprintf(message,"%.2f", data->v3_max);
    update_field_text(message, data->t_3v_max);
    // write 3V fail
    check_failure(message, *data->v3, data->v3_min, data->v3_max);
    update_field_text(message, data->t_f3v);

    // write 5V data
    sprintf(message,"%.2f", *data->v5);
    update_field_text(message, data->t_5v);
    // write 5V min
    sprintf(message,"%.2f", data->v5_min);
    update_field_text(message, data->t_5v_min);
    // write 5V max
    sprintf(message,"%.2f", data->v5_max);
    update_field_text(message, data->t_5v_max);
    // write 5V fail
    check_failure(message, *data->v5, data->v5_min, data->v5_max);
    update_field_text(message, data->t_f5v);

    // write 12V data
    sprintf(message,"%.2f", *data->v12);
    update_field_text(message, data->t_12v);
    // write 12V min
    sprintf(message,"%.2f", data->v12_min);
    update_field_text(message, data->t_12v_min);
    // write 12V max
    sprintf(message,"%.2f", data->v12_max);
```

```
update_field_text(message, data->t_12v_max);
// write 12V fail
check_failure(message, *data->v12, data->v12_min, data->v12_max);
update_field_text(message, data->t_f12v);

// write -12V data
sprintf(message,"%.2f", *data->vm12);
update_field_text(message, data->t_m12v);
// write -12V min
sprintf(message,"%.2f", data->vm12_min);
update_field_text(message, data->t_m12v_min);
// write -12V max
sprintf(message,"%.2f", data->vm12_max);
update_field_text(message, data->t_m12v_max);
// write m12V fail
check_failure(message, *data->vm12, data->vm12_min, data->vm12_max);
update_field_text(message, data->t_fm12v);

// write t1 data
sprintf(message,"%.1f", *data->t1);
update_field_text(message, data->t_t1);
// write t1 max
sprintf(message,"%.1f", data->t1_max);
update_field_text(message, data->t_t1_max);
// write t1 fail
check_failure(message, *data->t1, 0, data->t1_max);
update_field_text(message, data->t_ft1);

// write t2 data
sprintf(message,"%.1f", *data->t2);
update_field_text(message, data->t_t2);
// write t2 max
sprintf(message,"%.1f", data->t2_max);
update_field_text(message, data->t_t2_max);
// write t2 fail
check_failure(message, *data->t2, 0, data->t2_max);
update_field_text(message, data->t_ft2);

// write fan speed
if ( *data->fan_speed == 13 ) {
    sprintf(message,"%s","Baja");
} else if ( *data->fan_speed == 25 ) {
    sprintf(message,"%s","Media");
} else if ( *data->fan_speed == 37 ) {
    sprintf(message,"%s","Alta");
} else if ( *data->fan_speed == 50 ) {
```

```
        sprintf(message,"%s","Tope");
    } else {
        sprintf(message,"%s","Failure");
    }

    update_field_text(message, data->t_sfan);

}

void update_field_text(char* message, GtkWidget *field) {
  GtkTextBuffer *text_buffer;
  text_buffer = gtk_text_view_get_buffer(GTK_TEXT_VIEW(GTK_WIDGET(field)));

  gtk_text_buffer_set_text(text_buffer, message, -1);
}



int open_port(void)
{
  int fd_serie; /* File descriptor for the port */

  fd_serie = open("/dev/ttyACM0", O_RDWR | O_NOCTTY | O_NDELAY);
  if (fd_serie == -1)
  {
    /*
     * Could not open the port.
     */

    perror("open_port: Unable to open /dev/ttyACM1 - ");
  }
  else
    fcntl(fd_serie, F_SETFL, FNDELAY);

  //printf ( "In Open port fd_serie = %i\n", fd_serie);

  return (fd_serie);
}
```

headers.h

```
/* This function is used to check if a data value is inside its
 * working parameters, it returns a char with two spaces it the
 * value is inside its parameteres and FAILURE if it is not.
 */
void check_failure(char *message, float data, float min, float max);
```

```
/* This funcion executes when the window is close, it deataches
 * the shared memory and sends the end order to the child process.
 */
void on_window_destroy(GtkButton *object, Data_Structure *data);

/* This function splits the string from the microcontroller
 * into floats.
 */
void parse_data(Data_Structure *data);

/* This function is executed when the Togglebutton b_rec of the
 * gui is clicked. It set the child program into recording mode
 * when it is active and into waiting mode when it is not.
 */
void recording(GtkButton *object, Data_Structure *data);

/* Refresh the information display on the window with the data
 * process by the child thread.
 */
void refresh_window(GtkButton *object, Data_Structure *data);

/* Stop the refresh the information display on the window with
 * the data process by the child thread.
 */
void stop_refresh_window(GtkButton *object, Data_Structure *data);

/* This functions transforms data from the microcontroller values
 * to the real life numbers.
 */
void transform_data(Data_Structure *data);

/* This function refresh the information shown in the gui with the
 * information contained in the Data_Structure of the parent
 * process.
 */
void update_data(Data_Structure *data);

/* This function refresh the information show in a single text
 * field of the gui.
 */
void update_field_text(char* message, GtkWidget *field);
```

main.c

```
#include <fcntl.h>   /* File control definitions */
```

```
#include <errno.h>   /* Error number definitions */
#include <termios.h> /* POSIX terminal control definitions */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>
#include <sys/wait.h>
#include <string.h>
#include <locale.h>
//#include <fcntl.h>
//#include <errno.h>
//#include <termios.h>

#include "structures.h"
#include "headers.h"
#include "functions.h"

/*
 *  * 'open_port()' - Open serial port 1.
 *   *
 *    * Returns the file descriptor on success or -1 on error.
 *     */

//int main()
int main (int argc, char *argv[])
{
  const int shared_segment_size = 0x64;
  Data_Structure                *data;
  data = g_slice_new(Data_Structure); // Initialize widgets container.

  /* Initialize two shared memory segments */
  data->seg_order = shmget (IPC_PRIVATE, sizeof(int), IPC_CREAT
                            | IPC_EXCL | S_IRUSR | S_IWUSR);
  data->seg_req_data = shmget (IPC_PRIVATE, shared_segment_size,
                       IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
  data->seg_refreshing = shmget (IPC_PRIVATE, sizeof(int),
                       IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
  data->seg_v3 = shmget (IPC_PRIVATE, sizeof(float), IPC_CREAT
                            | IPC_EXCL | S_IRUSR | S_IWUSR);
  data->seg_v5 = shmget (IPC_PRIVATE, sizeof(float), IPC_CREAT
                            | IPC_EXCL | S_IRUSR | S_IWUSR);
```

```
data->seg_v12 = shmget (IPC_PRIVATE, sizeof(float), IPC_CREAT
                                | IPC_EXCL | S_IRUSR | S_IWUSR);
data->seg_vm12 = shmget (IPC_PRIVATE, sizeof(float), IPC_CREAT
                                | IPC_EXCL | S_IRUSR | S_IWUSR);
data->seg_t1 = shmget (IPC_PRIVATE, sizeof(float), IPC_CREAT
                                | IPC_EXCL | S_IRUSR | S_IWUSR);
data->seg_t2 = shmget (IPC_PRIVATE, sizeof(float), IPC_CREAT
                                | IPC_EXCL | S_IRUSR | S_IWUSR);
data->seg_fan_speed = shmget (IPC_PRIVATE, sizeof(float),
                    IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);

/* Initialize two pointers with the memory addresses
 * of the three memory sements, one in each pointer */
data->order = (int*) shmat(data->seg_order, (void*) 0, 0);
data->req_data = (char*) shmat(data->seg_req_data, (void*) 0, 0);
data->refreshing = (int*) shmat(data->seg_refreshing, (void*) 0, 0);
data->v3 = (float*) shmat(data->seg_v3, (void*) 0, 0);
data->v5 = (float*) shmat(data->seg_v5, (void*) 0, 0);
data->v12 = (float*) shmat(data->seg_v12, (void*) 0, 0);
data->vm12 = (float*) shmat(data->seg_vm12, (void*) 0, 0);
data->t1 = (float*) shmat(data->seg_t1, (void*) 0, 0);
data->t2 = (float*) shmat(data->seg_t2, (void*) 0, 0);
data->fan_speed = (float*) shmat(data->seg_fan_speed, (void*) 0, 0);

//Set limits.
data->v3_min = 3.135;
data->v3_max = 3.465;
data->v5_min = 4.75;
data->v5_max = 5.25;
data->v12_min = 11.4;
data->v12_max = 12.6;
data->vm12_min = 10.8;
data->vm12_max = 13.2;
data->t1_max = 55;
data->t2_max = 55;
//*data->fan_speed = 50;

/* This makes the child process start in the stand by mode */
*data->order = 0;

/* FORK */
data->pid = fork();
if(data->pid) {
  /* PARENT PROCES: This process creates the Gtk gui */
  #include "parent.h"
}
```

```
  else {
    /* CHILD PROCES: This process controls the transmissions
     * with the board and the data recording process */
    #include "child.h"
  }
  return 0;
}
```

parent.h

```
GtkBuilder              *builder;
GtkWidget               *window;

/* Initialize gtk objects */
gtk_init(&argc, &argv);
builder = gtk_builder_new();
gtk_builder_add_from_file(builder, "gui.xml", NULL);

/* Initialize data structure with all widget */
// GtkTextViews
data->t_t1 = GTK_WIDGET(gtk_builder_get_object(builder,"t_t1"));
data->t_t2 = GTK_WIDGET(gtk_builder_get_object(builder,"t_t2"));
data->t_ft1 = GTK_WIDGET(gtk_builder_get_object(builder,"t_ft1"));
data->t_ft2 = GTK_WIDGET(gtk_builder_get_object(builder,"t_ft2"));
data->t_fm12v = GTK_WIDGET(gtk_builder_get_object(builder,"t_fm12v"));
data->t_f12v = GTK_WIDGET(gtk_builder_get_object(builder,"t_f12v"));
data->t_f5v = GTK_WIDGET(gtk_builder_get_object(builder,"t_f5v"));
data->t_f3v = GTK_WIDGET(gtk_builder_get_object(builder,"t_f3v"));
data->t_3v = GTK_WIDGET(gtk_builder_get_object(builder,"t_3v"));
data->t_5v = GTK_WIDGET(gtk_builder_get_object(builder,"t_5v"));
data->t_m12v = GTK_WIDGET(gtk_builder_get_object(builder,"t_m12v"));
data->t_12v = GTK_WIDGET(gtk_builder_get_object(builder,"t_12v"));
data->t_3v_min = GTK_WIDGET(gtk_builder_get_object(builder,"t_3v_min"));
data->t_5v_min = GTK_WIDGET(gtk_builder_get_object(builder,"t_5v_min"));
data->t_12v_min = GTK_WIDGET(gtk_builder_get_object(builder,"t_12v_min"));
data->t_m12v_min = GTK_WIDGET(gtk_builder_get_object(builder,"t_m12v_min"));
data->t_3v_max = GTK_WIDGET(gtk_builder_get_object(builder,"t_3v_max"));
data->t_5v_max = GTK_WIDGET(gtk_builder_get_object(builder,"t_5v_max"));
data->t_12v_max = GTK_WIDGET(gtk_builder_get_object(builder,"t_12v_max"));
data->t_m12v_max = GTK_WIDGET(gtk_builder_get_object(builder,"t_m12v_max"));
data->t_t1_max = GTK_WIDGET(gtk_builder_get_object(builder,"t_t1_max"));
data->t_t2_max = GTK_WIDGET(gtk_builder_get_object(builder,"t_t2_max"));
data->t_sys_fail = GTK_WIDGET(gtk_builder_get_object(builder,"t_sys_fail"));
data->t_sfan = GTK_WIDGET(gtk_builder_get_object(builder,"t_sfan"));

// Bottons
```

```
data->b_rec = GTK_WIDGET(gtk_builder_get_object(builder,"b_rec"));
data->b_start = GTK_WIDGET(gtk_builder_get_object(builder,"b_start"));
data->b_stop = GTK_WIDGET(gtk_builder_get_object(builder,"b_stop"));
data->b_exit = GTK_WIDGET(gtk_builder_get_object(builder,"b_exit"));

/* Asign widgets to variables */
window = GTK_WIDGET(gtk_builder_get_object (builder, "window"));

/* Connect all signals using gtkbuilder */
gtk_builder_connect_signals(builder, data);

/* Show window */
gtk_widget_show(window);

/* Connects to the microcontroller and retrieves the data */
//---update_all(GTK_BUTTON(data->b_update_all), data);

/* Gtk infinite loop */
gtk_main();

return 0;
//waitpid(pid, &status, 0);  // BORRAR ANTES DE ENTREGAR.
```

structures.h

```
// This structure is use to store the data.
typedef struct _Data_Structure {
  int seg_order;
  int seg_req_data;
  int seg_refreshing;
  int seg_v3;
  int seg_v5;
  int seg_v12;
  int seg_vm12;
  int seg_t1;
  int seg_t2;
  int seg_fan_speed;

  int* order;
  char* req_data;
  int* refreshing;
  float* v3;
  float* v5;
  float* v12;
  float* vm12;
  float* t1;
```

```
float* t2;
float* fan_speed;

int pid;
float v3_min;
float v3_max;
float v5_min;
float v5_max;
float v12_min;
float v12_max;
float vm12_min;
float vm12_max;
float t1_max;
float t2_max;

/*
 * The next variables points to the GtkWidget of the same
 * names once they are initialize in parent process
 */
GtkWidget      *t_t1;
GtkWidget      *t_t2;
GtkWidget      *t_ft1;
GtkWidget      *t_ft2;
GtkWidget      *t_fm12v;
GtkWidget      *t_f12v;
GtkWidget      *t_f5v;
GtkWidget      *t_f3v;
GtkWidget      *t_3v;
GtkWidget      *t_5v;
GtkWidget      *t_m12v;
GtkWidget      *t_12v;
GtkWidget      *t_3v_min;
GtkWidget      *t_5v_min;
GtkWidget      *t_12v_min;
GtkWidget      *t_m12v_min;
GtkWidget      *t_3v_max;
GtkWidget      *t_5v_max;
GtkWidget      *t_12v_max;
GtkWidget      *t_m12v_max;
GtkWidget      *t_t1_max;
GtkWidget      *t_t2_max;
GtkWidget      *t_sys_fail;
GtkWidget      *t_sfan;
GtkWidget      *b_stop;
GtkWidget      *b_start;
GtkWidget      *b_exit;
```

```
  GtkWidget       *b_rec;
} Data_Structure;
```

C.2. Programa para el microcontrolador

A continuación se encuentra el código fuente del programa main.c para el microcontrolador.

```
// #includes
#include "main.h"
#include "VCP_F3.c"

// #defines
// #function prototypes

// #global variables
GPIO_InitTypeDef        GPIO_InitStructure;
RCC_ClocksTypeDef       RCC_Clocks;
GPIO_InitTypeDef        GPIO_InitStructure;

// Unused global variables that have to be included to ensure correct compiling
// ###### DO NOT CHANGE ######
__IO uint32_t   TimingDelay = 0;
__IO uint8_t    DataReady = 0;
__IO uint32_t   USBConnectTimeOut = 100;
__IO uint32_t   UserButtonPressed = 0;
__IO uint8_t    PrevXferComplete = 1;

// ++++++++++++++++++++++++++++++++ ADC CODE ++++++++++++++++++++++++++++++++
void ADCDelay(__IO uint32_t nTime);
__IO uint16_t         ADC1ConvertedValue = 0,
                      ADC1ConvertedVoltage = 0,
                      calibration_value = 0;
__IO uint32_t         ADCTimingDelay = 0;
ADC_InitTypeDef       ADC_InitStructure;
ADC_CommonInitTypeDef ADC_CommonInitStructure;
GPIO_InitTypeDef      GPIO_InitStructure;

// ++++++++++++++++++++++++++++++++ PWM CODE ++++++++++++++++++++++++++++++++
// function prototypes
void PWM_Init(void);

// global variables
GPIO_InitTypeDef        GPIO_InitStructure;
RCC_ClocksTypeDef       RCC_Clocks;
```

```
int main(void) {
  // Private variables
  uint16_t delay=500;
  char buf[40];
  uint16_t temp_max=0,temp1=0,temp2=0,volt3=0,volt5=0,volt12=0,voltm12=0;

  // Set the SysTick Interrupt to occur every 1ms)
  RCC_GetClocksFreq(&RCC_Clocks);
  if (SysTick_Config(RCC_Clocks.HCLK_Frequency / 1000))
    while(1);

  // Pull the USB D+ line low to disconnect the USB device from the PC
  VCP_ResetPort();
  Delay(500);
  // Initialize the STM32F3-Discovery as a Virtual COM Port
  // Device gets re-detected by the PC
  VCP_Init();

  // ++++++++++++++++++++++++++++++++ ADC CODE ++++++++++++++++++++++++++++++++
  // Configure the ADC clock
  RCC_ADCCLKConfig(RCC_ADC12PLLCLK_Div2);
  // Enable ADC1 clock
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_ADC12, ENABLE);

  // ADC Channel configuration
  // GPIOC Periph clock enable
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC, ENABLE);

  // Configure ADC Channel7 as analog input
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
  ADC_StructInit(&ADC_InitStructure);

  // Calibration procedure
  ADC_VoltageRegulatorCmd(ADC1, ENABLE);
  Delay(1);

  ADC_SelectCalibrationMode(ADC1, ADC_CalibrationMode_Single);
  ADC_StartCalibration(ADC1);

  while(ADC_GetCalibrationStatus(ADC1) != RESET );
  calibration_value = ADC_GetCalibrationValue(ADC1);

  ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
```

```
ADC_CommonInitStructure.ADC_Clock = ADC_Clock_AsynClkMode;
ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
ADC_CommonInitStructure.ADC_DMAMode = ADC_DMAMode_OneShot;
ADC_CommonInitStructure.ADC_TwoSamplingDelay = 0;
ADC_CommonInit(ADC1, &ADC_CommonInitStructure);

ADC_InitStructure.ADC_ContinuousConvMode = ADC_ContinuousConvMode_Enable;
ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
ADC_InitStructure.ADC_ExternalTrigConvEvent = ADC_ExternalTrigConvEvent_0;
ADC_InitStructure.ADC_ExternalTrigEventEdge = ADC_ExternalTrigEventEdge_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_OverrunMode = ADC_OverrunMode_Disable;
ADC_InitStructure.ADC_AutoInjMode = ADC_AutoInjec_Disable;
ADC_InitStructure.ADC_NbrOfRegChannel = 1;
ADC_Init(ADC1, &ADC_InitStructure);

// ADC1 regular channel7 configuration
ADC_RegularChannelConfig(ADC1, ADC_Channel_7, 1, ADC_SampleTime_7Cycles5);

// Enable ADC1
ADC_Cmd(ADC1, ENABLE);

//7 wait for ADRDY
while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_RDY));
// Start ADC1 Software Conversion
ADC_StartConversion(ADC1);

// +++++++++++++++++++++++++++++++ PWM CODE +++++++++++++++++++++++++++++++
uint16_t LED1dc=0;
// Set the SysTick Interrupt to occur every 1ms)
// CHECK IF NECCESARY
RCC_GetClocksFreq(&RCC_Clocks);
if (SysTick_Config(RCC_Clocks.HCLK_Frequency / 1000))
  while(1);
PWM_Init();

// ++++++++++++++++++++++++++++++ PINES CODE ++++++++++++++++++++++++++++++
// GPIOD Periph clock enable
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOE, ENABLE);

// Configure PE14 and PE15 in output pushpull mode
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15 | GPIO_Pin_14 | GPIO_Pin_13 |
                              GPIO_Pin_12 | GPIO_Pin_11 | GPIO_Pin_10 |
                              GPIO_Pin_9 | GPIO_Pin_8 | GPIO_Pin_7;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
```

```
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOE, &GPIO_InitStructure);

// LEDs test
// All on
delay = 1000;
GPIOE->BSRR = 0x0080; // PE.7 - Sistem
GPIOE->BSRR = 0x0100; // PE.8 - Temperature
GPIOE->BSRR = 0x0200; // PE.9 - -12V
GPIOE->BSRR = 0x0400; // PE.10 - 12V
GPIOE->BSRR = 0x1000; // PE.12 - 5V
GPIOE->BSRR = 0x4000; // PE.14 - 3,3V
Delay(2*delay);
// Set Off all LEDs in sequence
GPIOE->BRR = 0x0080; // PE.7 - Sistem
GPIOE->BRR = 0x0100; // PE.8 - Temperature
Delay(delay);
GPIOE->BRR = 0x0200; // PE.9 - -12V
GPIOE->BRR = 0x0400; // PE.10 - 12V
Delay(delay);
GPIOE->BRR = 0x1000; // PE.12 - 5V
GPIOE->BRR = 0x4000; // PE.14 - 3,3V
// Set Sistem LED On
Delay(delay);
GPIOE->BSRR = 0x0080; // PE.7 - Sistem
/* Main program loop */
while (1) {
  // ++++++++++++++++++++++++++++++ PWM CODE ++++++++++++++++++++++++++++++
  //Channel - set the duty
  TIM_SetCompare1(TIM4,LED1dc);
  //Channe2 - set the duty
  TIM_SetCompare2(TIM4,LED1dc);
  //Channe3 - set the duty
  TIM_SetCompare3(TIM4, LED1dc);
  delay=10;

  // 3,3V
  GPIOE->BRR = 0x0800; // A0: 0 (PE.11)
  GPIOE->BRR = 0x2000; // A1: 0 (PE.13)
  GPIOE->BRR = 0x8000; // A2: 0 (PE.15)
  Delay(delay);

  while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
  ADC1ConvertedValue =ADC_GetConversionValue(ADC1);
  while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
```

```
volt3 =ADC_GetConversionValue(ADC1);
if (volt3 > 2594 && volt3 < 2867) {
  // Set On +3,3V LED (PE.14)
  GPIOE->BSRR = 0x4000; // PE.14
} else {
  // Set Off +3,3V LED (PE.14)
  GPIOE->BRR = 0x4000; // PE.14
}

// 5V
GPIOE->BSRR = 0x0800; // A0: 1 (PE.11)
GPIOE->BRR = 0x2000; // A1: 0 (PE.13)
GPIOE->BRR = 0x8000; // A2: 0 (PE.15)
Delay(delay);

while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
ADC1ConvertedValue =ADC_GetConversionValue(ADC1);
while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
volt5 =ADC_GetConversionValue(ADC1);
if (volt5 > 2593 && volt5 < 2866) {
  // Set On +5V LED(PE.12)
  GPIOE->BSRR = 0x1000; // PE.12
} else {
  // Set Off +5V LED(PE.12)
  GPIOE->BRR = 0x1000; // PE.12
}

// 12V
GPIOE->BRR = 0x0800; // A0: 0 (PE.11)
GPIOE->BSRR = 0x2000; // A1: 0 (PE.13)
GPIOE->BRR = 0x8000; // A2: 0 (PE.15)
Delay(delay);

while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
ADC1ConvertedValue =ADC_GetConversionValue(ADC1);
while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
volt12 =ADC_GetConversionValue(ADC1);
if (volt12 > 2594 && volt12 < 2867) {
  // Set On +12V LED (PE.10)
  GPIOE->BSRR = 0x0400; // PE.10
} else {
  // Set Off +12V LED (PE.10)
  GPIOE->BRR = 0x0400; // PE.10
}

// -12V
```

```
GPIOE->BSRR = 0x0800; // A0: 0 (PE.11)
GPIOE->BSRR = 0x2000; // A1: 0 (PE.13)
GPIOE->BRR = 0x8000; // A2: 0 (PE.15)
Delay(delay);

while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
ADC1ConvertedValue =ADC_GetConversionValue(ADC1);
while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
voltm12 =ADC_GetConversionValue(ADC1);
if (voltm12 > 2448 && voltm12 < 2993) {
  // Set On -12V LED (PE.9)
  GPIOE->BSRR = 0x0200; // PE.9
} else {
  // Set Off -12V LED (PE.9)
  GPIOE->BRR = 0x0200; // PE.9
}

// temp1
GPIOE->BRR = 0x0800; // A0: 0 (PE.11)
GPIOE->BRR = 0x2000; // A1: 0 (PE.13)
GPIOE->BSRR = 0x8000; // A2: 0 (PE.15)
Delay(delay);

while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
ADC1ConvertedValue =ADC_GetConversionValue(ADC1);
while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
temp1 =ADC_GetConversionValue(ADC1);
temp_max = temp1;

// temp2
GPIOE->BSRR = 0x0800; // A0: 0 (PE.11)
GPIOE->BRR = 0x2000; // A1: 0 (PE.13)
GPIOE->BSRR = 0x8000; // A2: 0 (PE.15)
Delay(delay);

while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
ADC1ConvertedValue =ADC_GetConversionValue(ADC1);
while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
temp2 =ADC_GetConversionValue(ADC1);
if (temp2 > temp_max) {
  temp_max=temp2;
}

// Set PWM to power on the fans
if (temp_max < 1502) {
  // Fans at 25% capability
```

```
    LED1dc=13;
  } else if (temp_max < 1775){
    // Fans at 50% capability
    LED1dc=25;
  } else if (temp_max < 2048){
    // Fans at 75% capability
    LED1dc=37;
  } else {
    // Fans at 100% capability
    LED1dc=50;
  }

  // Temperature LED (PE.8)
  if (temp_max < 2185) {
    // Temperature under 55°C --> LED Off
    GPIOE->BRR = 0x0100; // PE.8
  } else {
    // Temperature over 55°C --> LED On
    GPIOE->BSRR = 0x0100; // PE.8
  }

  // Send all data through the USB.
  sprintf(buf, "%4u-%4u-%4u-%4u-%4u-%4u-%4u-\n", LED1dc,volt3,volt5,
                                                 volt12,voltm12,temp1,temp2);
  VCP_PutStr(buf);

  Delay(1000);
} // while(1)
} //main

// ++++++++++++++++++++++++++++++++ PWM CODE ++++++++++++++++++++++++++++++++
void PWM_Init(void) {
  GPIO_InitTypeDef        GPIO_InitStructure;
  TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
  TIM_OCInitTypeDef        TIM_OCInitStructure;

  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB, ENABLE);
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOD, ENABLE);
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

  // Pin configuration for PD14
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_14;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
```

```
  GPIO_Init(GPIOD, &GPIO_InitStructure);
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource14, GPIO_AF_2);

  // Time base configuration
  TIM_TimeBaseStructure.TIM_Prescaler = 71;
  TIM_TimeBaseStructure.TIM_Period = 49;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);

  // Output Control (OC) configuration - PWM1 Mode configuration: Channel1
  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;

  TIM_OC1Init(TIM4, &TIM_OCInitStructure);
  TIM_OC2Init(TIM4, &TIM_OCInitStructure);
  TIM_OC3Init(TIM4, &TIM_OCInitStructure);

  TIM_CtrlPWMOutputs(TIM4, ENABLE);
  TIM_Cmd(TIM4, ENABLE);
}
// ------------------------------- PWM CODE -------------------------------

// Function to insert a timing delay of nTime
// ###### DO NOT CHANGE ######
void Delay(__IO uint32_t nTime) {
  TimingDelay = nTime;

  while(TimingDelay != 0);
}

// Function to Decrement the TimingDelay variable.
// ###### DO NOT CHANGE ######
void TimingDelay_Decrement(void) {
  if (TimingDelay != 0x00) {
    TimingDelay--;
  }
}

// Unused functions that have to be included to ensure correct compiling
// ###### DO NOT CHANGE ######
// ============================================================================
uint32_t L3GD20_TIMEOUT_UserCallback(void) {
  return 0;
}

uint32_t LSM303DLHC_TIMEOUT_UserCallback(void) {
```

```
  return 0;
}
// ========================================================================
```